MASCHINELLE ERLERNUNG UND SIMULATION MORPHOLOGISCHER ABLEITUNGSREGELN

Inaugural-Dissertation
zur
Erlangung der Doktorwürde
der
Philosophischen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität
zu
BONN

vorgelegt von

KLAUS CHRISTIAN WOTHKE
aus
KÖLN

BONN 1985

Diss. Bonn
1984

MASCHINELLE ERLERNUNG UND SIMULATION MORPHOLOGISCHER ABLEITUNGSREGELN

Inaugural-Dissertation
zur
Erlangung der Doktorwürde
der
Philosophischen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität
zu
BONN

vorgelegt von

KLAUS CHRISTIAN WOTHKE
aus
KÖLN

BONN 1985

Angefertigt mit Genehmigung der Philosophischen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Referent: Prof. Dr. W. Lenders
2. Referent: Prof. Dr. J. Knobloch

Tag der mündlichen Prüfung: 12. Dezember 1984

Druck: Rheinische Friedrich-Wilhelms Universität Bonn

Meinem Vater,
Friedrich Wothke (1902-1972),
gewidmet

VORWORT

Die vorliegende Arbeit berichtet über die Ergebnisse eines Projektes, das ich in den Jahren 1981 bis 1984 am Institut für Kommunikationsforschung und Phonetik der Universität Bonn durchgeführt habe. Zweck des Vorhabens war Grundlagenforschung für ein anwendungsorientiertes Computerprogramm, das bestimmte linguistische Algorithmen maschinell erzeugen soll. Diese Algorithmen sollen zur maschinellen Bearbeitung von Teilbereichen der Derivationsmorphologie natürlicher Sprachen dienen. Ein solches Computerprogramm böte sich insbesondere als komfortables Werkzeug für Computerlinguisten, Linguisten und Informatiker an, da es den menschlichen Arbeitsaufwand bei der Erstellung bestimmter linguistischer Algorithmen bedeutend reduziert.

Zum behandelten Thema liegen meines Wissens noch keine anderen Arbeiten vor. Zwar waren einige Teilbereiche, mit denen die Arbeit in Berührung kommt, häufig Gegenstand linguistischer und computerlinguistischer Arbeiten, jedoch wurden sie unter anderen Fragestellungen angegangen und entsprechend anders strukturiert. Aus diesen Gründen war es erforderlich, an vielen Stellen dieser Arbeit eine eigene Terminologie zu verwenden, die meiner Fragestellung und der sich daraus ergebenden speziellen Strukturierung des Objektbereiches entspricht. Diese Einführung neuer, dem Gegenstand angemessener Terminologie garantiert eher die inhaltliche Eindeutigkeit der Darstellung. Allerdings hat sie den Nachteil, daß die Lektüre etwas erschwert wird. Ich bitte, dies zu entschuldigen.

Besonderen Dank für Anregungen zu dieser Arbeit schulde ich allen Mitarbeitern der Abteilung Linguistische Datenverarbeitung

am Institut für Kommunikationsforschung und Phonetik der Universität Bonn.

Bonn, im Sommer 1984

Klaus Wothke

INHALTSVERZEICHNIS

Vorwort.....V
Abkürzungsverzeichnis.....IX
Symbolverzeichnis.....XI

1 Einleitung.....1
1.1 Thematische Eingrenzungen.....1
1.2 Überblick über den Aufbau der Arbeit.....11

2 Der Forschungsstand.....14
2.1 Arbeiten zur maschinellen Erlernung oder Simulation derivativer Wortbildungsregeln.....16
2.1.1 Bünting.....17
2.1.2 Krallmann/Krumnack/Soeffner.....22
2.1.3 Pacak/Pratt.....27
2.1.4 ISLIB.....32
2.1.5 JUDO.....37
2.2 Lernalgorithmen für andere sprachliche Substitutionsbeziehungen.....40
2.2.1 Wolf.....41
2.2.2 Oakey/Cawthorn.....51
2.2.3 PELIKAN.....57

3 Problemstellung und Hypothesen.....61
3.1 Problemstellung.....61
3.2 Hypothesen.....84
3.3 Exkurs: Zum Modellcharakter des Lernalgorithmus sowie zu seinem Bezug zu empiristischen und rationalistischen Spracherwerbstheorien.....88

4 Überblick über das Programmsystem PRISM.....91

5 Die Prozeduren des Programmsystems.....97
5.1 MONITOR.....98
5.2 GETOPTN.....99
5.3 CHKCRPS.....101
5.4 DISCOV.....102
5.5 STMTOUT.....124
5.6 TODSET.....126
5.7 FRODSET.....127
5.8 APPLY.....128
5.9 DERIVE.....129
5.10 LISTOUT.....132
5.11 INVERT.....132
5.12 LLTMTCH.....133

6 Wortpaarlisten für die Evaluierung von PRISM.....135
6.1 Gründe für die Wahl der betrachteten Ableitungsbeziehungen.....135
6.2 Beschreibung der zugrunde gelegten Wortpaarlisten.....137

7 Hypothesenüberprüfung.....144

8 Anwendungsmöglichkeiten von PRISM.....176

9 Unzulänglichkeiten und Weiterentwicklungsmöglichkeiten von PRISM.....181

10 Anhang.....188
10.1 PRISM-Quellprogramm.....188
10.2 Ergebnisse der Anwendung von PRISM auf die Motion im Französischen.....272
10.3 Ergebnisse der Anwendung von PRISM auf die Negation englischer Adjektive.....296

Literaturverzeichnis.....306

Sachregister.....311

ABKÜRZUNGSVERZEICHNIS

Allgemein übliche oder im Text erklärte Abkürzungen sind nicht aufgeführt.

abs.	- absolut(e)
ADJ	- Adjektiv
AI	- Artificial Intelligence
allg.	- allgemein(e)
Cal.	- California
CPU	- central processing unit (Zentraleinheit einer elektronischen Rechenanlage)
Def.	- Definition
Diss.	- Dissertation
EBCDIC (auch: EBCDI-Code)	- extended binary coded decimal interchange code (Standardisierter Binärkode zur Kodierung von Daten für die Verarbeitung durch eine elektronische Rechenanlage)
e. g.	- exempli gratia (= zum Beispiel)
engl.	- englisch
erw.	- erweiterte
frz.	- französisch
GB	- Great Britain
gdw.	- genau dann wenn
GMD	- Gesellschaft für Mathematik und Datenverarbeitung
Habil.-Schr.	- Habilitationsschrift
IBM	- international business machines (= US-amerikanischer Konzern)
i. e.	- id est (= das heißt)
IKP	- Institut für Kommunikationsforschung und Phonetik der Universität Bonn
IPK	- Institut für Phonetik und Kommunikationsforschung der Universität Bonn
i. U.	- im Unterschied
KI	- Künstliche Intelligenz
LDV	- Linguistische Datenverarbeitung
Mass.	- Massachusetts
M.I.T.	- Massachusetts Institute of Technology
N	- Substantiv
N. J.	- New Jersey

N. Y.	- New York
P.	- page (engl.: Seite)
P. C.	- per cent (engl.: Prozent)
Prep.	- preparation (engl.: Vorbereitung)
proz.	- prozentual
s. a.	- see above (engl.: siehe oben)
überarb.	- überarbeitete
verb.	- verbesserte
Verf.	- Verfassers
Vol.	- volume (engl.: Band)
vollst.	- vollständig

SYMBOLVERZEICHNIS

Allgemein übliche Symbole sind nicht aufgeführt.

Die in Flußdiagrammen verwendeten Sinnbilder sind erläutert in: DIN 66001.

Eine Beschreibung der in Nassi-Shneiderman-Diagrammen verwendeten Symbole findet sich in: Bauknecht/Zehnder (1980: 121f.).

Zur Erläuterung der Symbole des internationalen phonetischen Alphabets vergleiche man: The principles of the international phonetic association. London 1949.

'...'	- "... ist eine objektsprachliche Zeichenkette." Der sog. Nullstring, d. h. die Zeichenkette der Länge 0, wird mit '' notiert.
*'...'	- "Die Zeichenkette ... gehört nicht der betrachteten Sprache an."
{...}[1]	- "... ist ein objektsprachliches Morphem."
/.../	- "... ist die phonemische Transkription eines objektsprachlichen Ausdrucks."
<...>	- "... ist Bedeutungsangabe zu der vor <...> stehenden objektsprachlichen Zeichenkette."
#	- Wortgrenze
'...' → '---'	- "Ersetze die Zeichenkette ... durch die Zeichenkette ---." (= Substitutionsmuster)
$X \rightarrow Y / \#__ (Z_1 \mid \ldots \mid Z_i \mid \ldots \mid Z_n)$	- "Ersetze die Zeichenkette X durch die Zeichenkette Y, falls X am Wortanfang steht, und zwar unmittelbar links von Z_1 oder ... oder Z_i oder ... oder Z_n." (= Präfixorientierte Substitutionsanweisung).

1 {...} bezeichnet das Morphem ..., während {'...'} die Menge bezeichnet, die als Element die Zeichenkette ... enthält.

$X \rightarrow Y/(Z_1 \mid \ldots \mid Z_i \mid \ldots \mid Z_n)\ __\#$	- "Ersetze die Zeichenkette X durch die Zeichenkette Y, falls X am Wortende steht, und zwar unmittelbar rechts von Z_1 oder ... oder Z_i oder ... oder Z_n." (= Suffixorientierte Substitutionsanweisung)
$\#\ X\neg$	- "Beginnt eine Wortform mit der Zeichenkette X, so erzeuge keine Ableitung zu der Wortform." (= Präfixorientierte Blockierungsanweisung)
$X\ \#\neg$	- "Endet eine Wortform mit der Zeichenkette X, so erzeuge keine Ableitung zu der Wortform." (= Suffixorientierte Blockierungsanweisung)
'...' <==> '---'	- "Die Zeichenkette ... steht zu der Zeichenkette --- in einer morphologischen Ableitungsbeziehung."
... <==> ---	- Die Menge all derjenigen geordneten Wortpaare, deren Erstkomponenten Vertreter der Wortart ... sind und deren Zweitkomponenten diejenigen Vertreter der Wortart --- sind, die zu den Erstkomponenten in einer morphologischen Ableitungsbeziehung stehen.
'...' || '---'	- "Konkateniere die Zeichenkette ... mit der Zeichenkette ---."
... || ---	- "Konkateniere Zeichenketten der Kategorie ... mit Zeichenketten der Kategorie ---."
... := ---	- "... ist nach Definition gleich ---."
... >= ---	- "... ist größer oder gleich ---."
... <= ---	- "... ist kleiner oder gleich ---."
$\sum_{i=1}^{n} x_i$	- Summe der Werte x_i für i = 1 bis n.
(x_1, x_2)	- Geordnetes Paar mit der Erstkomponente x_1 und der Zweitkomponente x_2.
{x, y, ...}	- Menge der Elemente x, y, ...

$A \subset B$	- "Die Menge A ist echte Teilmenge der Menge B."
$\bigcup_{i=1}^{n} M_i$	- Vereinigungsmenge aller Mengen M_i für i = 1 bis n.
$\neg \ldots$	- "Nicht ..." (= Negator)
$\ldots \ \& \ ---$	- "... und ---" (= Konjunktor)
$\ldots \mid ---$	- "... oder ---" (= Disjunktor)
$\ldots \supset ---$	- "wenn ..., so ---" (= Implikator)
$\ldots \Leftarrow ---$	- "... erhält den Wert ---"

1 EINLEITUNG

1.1 Thematische Eingrenzungen

Gegenstand der Arbeit

Gegenstand dieser Arbeit ist die Entwicklung, Implementierung und Evaluierung eines experimentellen Algorithmus, der derivationsmorphologische Regeln zu beliebigen Sprachen erlernt und anwendet. Zum Lernen der Regeln sollen dem Algorithmus geeignete Beispiele vorgegeben werden. Die Beispiele sollen Mengen geordneter Wortpaare sein, zwischen deren Erst- und Zweitkomponenten eine morphologische Ableitungsbeziehung besteht. Eine solche Menge wäre etwa für die Negation englischer Adjektive

('accessible' <zugänglich>, 'inaccessible' <unzugänglich>),
('cautious' <vorsichtig>, 'incautious' <unvorsichtig>),
('mature' <reif>, 'immature' <unreif>),
('perfect' <vollkommen>, 'imperfect' <unvollkommen>).

Zu einer vorgegebenen Beispielmenge soll der Algorithmus eine Menge von Regeln erlernen, mittels derer er die Zweitkomponenten der Paare aus den jeweiligen Erstkomponenten erzeugen kann. Der Algorithmus soll die Regeln allerdings so konstruieren, daß sie ihm gestatten, nicht nur zu den Erstkomponenten der Beispielmenge die entsprechenden Zweitkomponenten zu bilden, sondern auch zu möglichst vielen weiteren Wörtern, die der gleichen Gruppe von Wörtern angehören wie die vorgegebenen Erstkomponenten. Die Regeln sollen dabei ausschließlich auf der graphischen[1] Gestalt

1 Wir verwenden in dieser Arbeit die Termini '*graphisch*' und '*Graph*' und nicht - wie es in anderen Arbeiten in diesem Zusammenhang häufig fälschlich der Fall ist - die Termini 'graphemisch' und 'Graphem'. In schriftlichen Äußerungen, die ja Bestandteil der parole sind, können nämlich keine Grapheme auftreten, sondern nur ihre Realisierungen, die Graphen, da Grapheme abstrakte Einheiten des Sprachsystems, der langue, sind. (Nähere Informationen zum hier zugrunde gelegten Verständnis des Verhältnisses von Graph und Graphem findet man bei Pulgram (1951).

der Wörter operieren. Ausgehend von obiger Beispielmenge sollten also etwa folgende Regeln zur Negation englischer Adjektive gebildet werden:

1. Beginnt die Wortform mit 'm' oder 'p', so konkateniere 'im' mit der Wortform.
2. Ist Regel 1 nicht anwendbar, so konkateniere 'in' mit der Wortform.

Diese Regeln würden dem Algorithmus erlauben, nicht nur zu den Erstkomponenten der angegebenen Beispielmenge die korrekten Negationen zu erzeugen, sondern auch zu den meisten anderen englischen Adjektiven, soweit sie nicht mit den Präfixen 'dis' oder 'un' negiert werden. So gestatten die Regeln z. B. auch die korrekte Bildung von

'inapplicable'	<unanwendbar>	zu	'applicable'	<anwendbar>,
'incomplete'	<unvollständig>	zu	'complete'	<vollständig>,
'immoral'	<unmoralisch>	zu	'moral'	<moralisch>,
'implausible'	<unglaubwürdig>	zu	'plausible'	<glaubhaft>,
'invariable'	<unveränderlich>	zu	'variable'	<veränderlich>

Dies ist allerdings nur möglich, wenn der Lernmechanismus des Algorithmus über gewisse Analogiebildungs- und Generalisierungsstrategien verfügt. Sie erlauben es ihm erst, die Regeln so allgemein zu formulieren, daß er auch zu Erstkomponenten, die nicht zum Lernen vorgegeben wurden, korrekte Zweitkomponenten erzeugen kann. Neben diesem Vorteil bergen Generalisierungen und Analogiebildungen auch die Gefahr der *Übergeneralisierung* in sich, d. h. es besteht die Gefahr, daß Regeln zu allgemein formuliert werden so daß sie auf zu viele Erstkomponenten anwendbar sind und inkorrekte Zweitkomponenten erzeugt werden. Im Beispiel macht sich eine Übergeneralisierung der erlernten Regeln bei ihrer Anwendung auf die Adjektive 'legible' <leserlich> und 'regular' <regelmäßig> bemerkbar. Anstelle der korrekten Negationen 'illegible' <unleserlich> und 'irregular' <unregelmäßig> würden hier mittels Regel 2 die fehlerhaften Formen *'inlegible' und *'inregular' gebildet. Für diesen Mangel sind 2 Gründe verantwortlich:

1. Regel 2 ist zu allgemein.
2. Die vorgegebene Beispielmenge erlaubt dem Algorithmus nicht die Erlernung von Regeln, die Adjektiv-Negationen mit den

{in}-Allomorphen /ir/ und /il/ bilden, und zwar einfach weil Wortpaare zu diesen Bildungsmustern in der Beispielmenge fehlen. (Dieser zweite Grund läßt sich durchaus als Ursache für den ersten Grund einstufen.)

Der entgegengesetzte Fehler zur Übergeneralisierung ist die sogenannte *Überdiskriminierung*. Sie entsteht, wenn bei der Konstruktion von Regeln nicht ausreichend generalisiert wird und die Regeln somit zu speziell formuliert werden, so daß sie auf zu wenige Erstkomponenten anwendbar sind. Eine Überdiskriminierung läge beispielsweise vor, wenn ausgehend von der anfangs genannten Beispielmenge folgende Regel anstelle der oben angeführten 1. Regel konstruiert würde:

1'. Beginnt die Wortform mit 'ma' oder 'pe', so konkateniere 'im' mit der Wortform.

Würde diese Regel in Verbund mit der obigen 2. Regel auf 'moral' und 'plausible' angewendet, so würden nicht die korrekten Negationen 'immoral' und 'implausible' gebildet. Vielmehr entstünden die fehlerhaften Negationen *'inmoral' und *'inplausible', weil die zu spezielle Regel 1' weder auf 'moral' noch auf 'plausible' anwendbar ist und somit Regel 2 zum Zuge käme.

Wesentliches Teilziel dieser Arbeit wird es sein, den Algorithmus weitestgehend mit Analogiebildungs- und Generalisierungsstrategien auszustatten, ihn aber gleichzeitig - soweit dies möglich ist - vor Übergeneralisierungen und vor Überdiskriminierungen zu schützen.

Als Basismodell bietet sich für den zu konstruierenden Algorithmus folgende Gliederung in eine Lernkomponente, eine Wissensbasis und eine Anwendungskomponente an (vgl. Abb. 1.1):

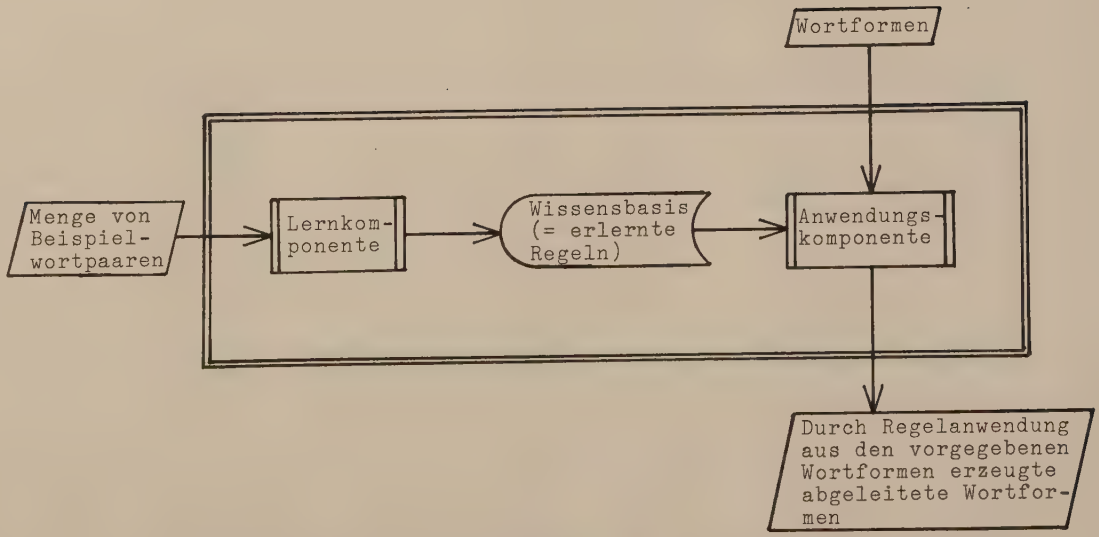


Abb 1.1: Basismodell für einen Algorithmus zur Erlernung und Simulation morphologischer Ableitungsregeln.

Die *Lernkomponente* soll aus einer vorgegebenen Menge von Beispielwortpaaren eine Menge von Regeln ableiten und in der Wissensbasis ablegen.

Die *Wissensbasis* soll als langfristiger Speicher des Gesamtsystems dienen. Sie soll beliebig viele, von der Lernkomponente generierte Regelmengen für verschiedene morphologische Ableitungsbeziehungen aus beliebigen natürlichen Sprachen aufnehmen können. Die Regeln sollen so strukturiert sein, daß sie effizient maschinell ausgeführt werden können. Dies bedeutet u. a., daß sie in einer formalen Sprache formuliert werden müssen und nicht in natürlicher Sprache, wie wir es oben für die Regeln zur Negation englischer Adjektive getan hatten.

Die *Anwendungskomponente* soll Regelmengen, die von der Lernkomponente erzeugt und in der Wissensbasis abgelegt wurden, auf vorgegebene Wortformen anwenden und dadurch Wortformen erzeugen, die zu den vorgegebenen Wortformen in einer morphologischen Ableitungsbeziehung stehen. Die Art der morphologischen Ableitungsbeziehung wird durch die jeweils angewendete Regelmenge festgelegt.

Neben diesem von uns favorisierten Basismodell sind noch andere Grundkonzeptionen eines Algorithmus zur Erlernung und Simulation morphologischer Ableitungsregeln denkbar. Eine wesentliche Alternative werden wir in Abschnitt 9 vorstellen.

Abschließend sei betont, daß die obige Darstellung des Gegenstandes dieser Arbeit eine *vorläufige* Eingrenzung ist. Erst in Abschnitt 3 werden wir eine systematische und detaillierte Ausarbeitung der Problemstellung und der Hypothesen liefern.

Erkenntnisinteresse und Anwendungsbezüge

Es handelt sich bei dieser Arbeit um *Grundlagenforschung* für einen *intelligenten anwendungsorientierten* Algorithmus. Unser primäres Erkenntnisinteresse gilt dabei den Strategien, mit denen der Algorithmus ausgestattet werden muß, um Regeln erlernen zu können. Mit der Verwendung des Attributs 'intelligent' verbinden wir *nicht* den Anspruch, daß dieser Algorithmus bestimmte menschliche Verhaltensweisen modelliert. Vielmehr nehmen

wir die in der Künstlichen Intelligenz-Forschung vorherrschende Position ein und verwenden dieses Attribut, um zu verdeutlichen, daß der angestrebte Algorithmus bestimmte Verhaltensweisen zeigen soll, die Menschen zwar als intelligent einstufen, die jedoch nicht unbedingt mit menschlichen Verhaltensweisen übereinstimmen. Diese Position charakterisieren namhafte Vertreter der Künstlichen Intelligenz-Forschung folgendermaßen:

> Artificial Intelligence "was an attempt to build intelligent machines without any prejudice toward making the system simple, biological or *humanoid* (Hervorhebung des Verf.)." (Minsky 1968a: 7)

> "AI does not require that an intelligent program demonstrate *human* intelligence (...; Auslassung des Verf.)". (Cohen/ Feigenbaum 1982: 7)

Es ist im übrigen auch fraglich, ob bei der Konstruktion *anwendungsorientierter* Systeme das Bestreben, menschliche kognitive Prozesse möglichst genau nachzubilden, immer zur effektivsten Lösung führen würde. So zeigt z. B. Wolfgang Krause (1982), ein Vertreter der Linguistischen Datenverarbeitung, an einer Reihe von Beispielen, daß die Simulation kognitiver Prozesse nicht als Allheilmittel für Mängel maschineller Sprachverarbeitungssysteme angesehen werden kann. Er macht deutlich, daß der Simulationsansatz zwar in manchen Bereichen zu brauchbaren Lösungen führen kann, daß jedoch aus Effektivitätsgründen in anderen Teilbereichen nicht-simulative Lösungen vorzuziehen sind. Zusammenfassend führt Krause (1982: 63) aus:

> "Und deshalb kann eine deckungsgleiche Simulation der menschlichen Sprachverarbeitung nicht in allen Teilbereichen zur bestmöglichen maschinellen Sprachverarbeitung führen, weil Computer 'natürlich' anders strukturiert sind als menschliche Gehirne.
>
> Dies spricht vom *Effektivitätsstandpunkt* aus gegen eine deckungsgleiche Simulation. Die tiefgreifenden Strukturunterschiede zwischen den Prozeßabläufen von Mensch und Computer müssen berücksichtigt werden (Hervorhebung des Verf.)."

Folgende *Anwendungsbereiche* haben wir für den angestrebten Algorithmus im Auge:

Wissenschafts*intern* könnte er verwendet werden

- zur maschinellen Erstellung von Algorithmen für die automatische Generierung morphologischer Ableitungen,

- als technisches Hilfsmittel bei der Erstellung einer deskriptiven Grammatik der Derivationsmorphologie,
- sowie zur Messung der Regelhaftigkeit morphologischer Ableitungsbeziehungen auf graphischer Ebene.

Wissenschafts*extern* ließe sich das Verfahren z. B. innerhalb linguistisch orientierter maschineller Indexierungssysteme einsetzen, um abgeleitete Wörter zu generieren: Wenn dort z. B. die potentiellen Deskriptoren, die das System zu einem Text liefern soll, Substantive sein müssen, jedoch wesentliche Inhalte des Textes auch in Form von Adjektiven realisiert sind, so könnte *zunächst* die Lernkomponente ausgehend von einer Adjektiv-Substantiv-Liste eine Menge von Regeln zur Substantivierung von Adjektiven erstellen. *Nach Abschluß* der Lernphase könnte die Anwendungskomponente durch Anwendung der erlernten Regeln auf die - zuvor mit einem anderen Programm identifizierten und lemmatisierten - Adjektive eines zu indexierenden Textes die entsprechenden potentiellen Deskriptoren in Form von Substantiven erzeugen.

Gegenüber nicht-lernenden Systemen, die z. Z. vorherrschen, ist ein solches, lernendes System beim Einsatz in einem linguistisch orientierten Indexierungssystem wesentlich wartungsfreundlicher:

Bei einem nicht-lernenden System müssen Modifizierungen der Regelmengen vom Benutzer, bzw. Programmierer durchgeführt werden. Die dabei zu bearbeitenden Regelmengen dürften so umfangreich und unüberschaubar sein, daß

- erstens der Arbeitsaufwand bei einer intellektuellen Modifizierung äußerst hoch ist und
- zweitens leicht Folgefehler bei einer Modifizierung auftreten.

Weiterhin müssen die zu ergänzenden Regeln vom Benutzer selbst formuliert werden.

Ein lernendes System dagegen

- reduziert den Arbeitsaufwand des Benutzers auf die Eingabe von Beispielen, aus denen gelernt werden soll,
- und es kann automatisch Konsistenzprüfungen bei der Erstellung von Regelmengen durchführen.

Bei einigen der angeführten Anwendungsmöglichkeiten ist vielleicht zweifelhaft, ob die Verwendung eines mit *Regeln* arbeitenden Verfahrens für die Erzeugung abgeleiteter Wörter überhaupt sinnvoll ist, und man wird möglicherweise einem *lexikon*basierten Verfahren den Vorzug geben wollen. Ein solches Verfahren würde mit einem maschinenlesbaren Lexikon arbeiten, das als Einträge Wortformen mit ihren Ableitungen enthält. Soll das Verfahren zu einer vorgegebenen Wortform eine Ableitung bereitstellen, so muß es im Lexikon lediglich den Eintrag zu der Wortform aufsuchen und die dort verzeichnete Ableitung ausgeben. Auf die Frage, welcher der beiden Ansätze der bessere ist, läßt sich keine generelle Antwort geben. Beide Ansätze haben spezifische Vor- und Nachteile, die auch nur in bestimmten Situationen zum Tragen kommen:

- *Lexikonbasierte* Verfahren sind insbesondere dann zu wählen, wenn eine morphologische Ableitungsbeziehung wenige oder keine Regelhaftigkeiten aufweist und wenn ein hoher Exaktheitsgrad bei der Bereitstellung abgeleiteter Wörter erforderlich ist. Ein Nachteil ist der hohe Arbeitsaufwand bei der Erstellung des Lexikons. Weiterhin versagt das Verfahren für alle Wortformen, die nicht mit ihren Ableitungen im Lexikon verzeichnet sind, also insbesondere bei Neubildungen.
- Ein *regelbasiertes* Verfahren ist dagegen vorzuziehen, wenn die morphologische Ableitungsbeziehung sehr regelhaft ist. Es hat dann gegenüber einem Lexikon die Vorzüge, daß es mit geringerem Arbeitsaufwand bereitgestellt werden kann und daß es auch korrekte Neubildungen erzeugen kann, die zum Zeitpunkt als die Regeln des Verfahrens erstellt wurden, noch nicht in Gebrauch waren. Es kann somit i. U. zu einem Lexikon auch der Produktivität derivativer Wortbildungen Rechnung tragen. Wesentlicher Nachteil des regelbasierten Ansatzes ist allerdings, daß er infolge Übergeneralisierung und Überdiskriminierung in der Regel einen gewissen Prozentsatz fehlerhafter Ableitungen einschließt.

Da in einer Sprache die Bildung abgeleiteter Wörter selten völlig unregelmäßig und auch selten durchgehend regelhaft ist, wird in der Praxis i. allg. eine Mischung aus lexikon- und regelbasierter

Verfahren zu empfehlen sein: Die Ausnahmen sollten vom Lexikon geliefert werden. Regelmäßige Ableitungen sollte der Regelteil des Verfahrens bereitstellen. Trotz dieser gegenseitigen Ergänzung von Lexikon und Regelteil werden wir uns in dieser Arbeit ausschließlich mit der maschinellen Erlernung des Regelteils befassen. Wo bei konkreten Anwendungen die Schnittstelle zwischen unserem Algorithmus und einer Lexikonaufsuche gelegt werden könnte, werden wir in Abschnitt 9 darlegen.

Relevante Typen des Lernens

Abschließend seien kurz die Arten des Lernens näher charakterisiert, mit denen wir es in den folgenden Abschnitten zu tun haben werden. Es handelt sich dabei um das inferentielle Lernen, das Lernen durch Analogieschluß und das Regellernen:

- Unter *inferentiellem Lernen* verstehen wir den Prozess der Gewinnung zusätzlichen Wissens aus bereits vorhandenem Wissen mittels Induktion und/oder Deduktion. (Das zusätzliche Wissen ist in dem vorhandenen Wissen bereits implizit angelegt, wird jedoch durch den Prozess des inferentiellen Lernens explizit gemacht.)

  Klix (1982: 387f.) führt folgendes Beispiel für deduktives inferentielles Lernen an:

  "A kind of deductive reasoning was recognized by Gillan (1981; Ergänzung des Verf.) in the following context: Behind a wall, invisible for the animals, there are two hiding places for food, say p and q (...; Auslassung des Verf.). Two young chimpanzees, say A and B, observe how food seems to be carried to the places. A is allowed to go in the one direction. It prefers direction p. Having seen this direction, chimpanzee B is removed from the scene. It becomes returned when A is coming back with a fruit in its hand. If allowed, B starts immediately in direction of q. The cognitive background of such a behavior decision is like a disjunctive syllogism:

  $$(p \mid q) \supset (\neg p \supset q)$$

  It is use of knowledge, but in a revised, indirect mode. Inferences allow to derive implicitly stored knowledge."

- Beim *Lernen durch Analogieschluß* wird von Ähnlichkeit oder Identität zweier oder mehrerer Objekte oder Sachverhalte in bestimmten Eigenschaften auf die Ähnlichkeit oder Identität dieser Objekte oder Sachverhalte in weiteren Eigenschaften

geschlossen. Das Lernen durch Analogieschluß kann neben dem induktiven und dem deduktiven Lernen auch als dritte Untergruppe des inferentiellen Lernens klassifiziert werden.

Lernen durch Analogieschluß (in Verbindung mit Generalisierung und Regellernen (s. u.)) liegt beispielsweise vor, wenn ein Lerner aus den Paaren

('table' <Tisch>, 'tables' <Tische>)
('boy' <Junge>, 'boys' <Jungen>)

englischer Substantive im Singular und im Plural schließt, daß auch zu den übrigen englischen Substantiven im Singular, beispielsweise zu 'proof' <Beweis>, 'pencil' <Bleistift> und 'car' <Auto> der Plural durch Anhängen von 's' gebildet wird. Daß die Gefahr von Übergeneralisierungen bei Analogieschlüssen besteht, wird bereits dadurch deutlich, daß der Lerner fälschlicherweise auch zu 'thief' <Dieb>, 'lady' <Dame> und 'goose' <Gans> den Plural durch Anhängen von 's' bilden würde.

- '*Regellernen*' schließlich bezeichnet den Prozess der Gewinnung prozeduralen Wissens aus statischem Wissen.

Ein Beispiel für Regellernen haben wir bereits bei der Erläuterung des Lernens durch Analogieschluß gegeben. Dort wurde in Verbindung mit einem Analogieschluß die (übergeneralisierende) Regel

'Der Plural englischer Substantive wird durch Anhängen von 's' an die Singularform gebildet.'

erlernt.

Die tatsächliche Existenz des Regellernens wird offensichtlich, wenn man sich vor Augen führt, daß jeder Sprecher einer Sprache nach Abschluß des Spracherwerbs in der Lage ist abgeleitete Wörter und Sätze zu äußern, die er zuvor nie gehört hat, die jedoch grammatisch korrekt sind. Ohne das Erlernen von Regeln wäre der Sprecher lediglich fähig, bereits wahrgenommene Ableitungen und Sätze zu äußern (vgl. dazu z. B. Klix 1982: 388).

Diese 3 Arten des Lernens müssen keineswegs so disjunkt voneinander auftreten, wie es vielleicht ihre getrennte Erläuterung

suggeriert. So dürfte etwa das Regellernen häufig mit dem inferentiellen Lernen und dem Lernen durch Analogieschluß eng verzahnt sein.

Neben den 3 genannten Typen des Lernens werden in der Psychologie noch weitere Typen des Lernens unterschieden, mit denen wir im folgenden jedoch nicht in Berührung kommen werden. So nennt etwa F. Klix (1982: 389) neben den 3 angeführten noch 4 weitere Typen des Lernens:

- Lernen durch Gewöhnung,
- Lernen durch klassische Konditionierung,
- Lernen durch instrumentelle (operante) Konditionierung,
- Lernen durch Hypothesenbildung.

Es sei hier nochmals nachdrücklich davor gewarnt, aus unserer Bezugnahme auf Typisierungen des Lernens in der Psychologie fälschlicherweise den Schluß zu ziehen, es ginge in der vorliegenden Arbeit um die Modellierung menschlicher Lernprozesse. Wie oben bereits ausgeführt, trifft dies nicht zu. Vielmehr handelt es sich hier ausschließlich um Grundlagenforschung für ein anwendungsorientiertes Programmsystem. Da einige Verhaltensweisen des Programmsystems bestimmten menschlichen Verhaltensweisen ähneln, boten sich zur Bezeichnung und Charakterisierung dieser Verhaltensweisen des Programms die Bezeichnungen und Charakterisierungen für die entsprechenden menschlichen Verhaltensweisen an, so etwa die Bezeichnungen 'Erlernung', 'Generalisierung' und 'Analogieschluß'.

1.2 Überblick über den Aufbau der Arbeit

Abschnitt 2 gibt eine Darstellung des Forschungsstandes zum Gegenstand dieser Arbeit. Zunächst werden wir in seinem *Unterabschnitt 2.1* existierende Arbeiten zur maschinellen Erlernung oder Simulation derivativer Wortbildungsregeln behandeln. Diese Projekte sind für uns aus verschiedenen Gründen von Interesse:

- Sie können uns Hinweise geben für die Wahl einer geeigneten formalen Repräsentationssprache zur Darstellung derivativer Wortbildungsregeln.
- Sie ermöglichen uns eine vergleichende Bewertung des zu kon-

struierenden Algorithmus.

- Ihre Schwachstellen zeigen uns, welche Teile bei der Konstruktion unseres Verfahrens besondere Beachtung verdienen müssen.

Wie sich in 2.1 zeigen wird, existieren zwar einige Arbeiten zur maschinellen *Simulation* derivativer Wortbildungsregeln, jedoch praktisch keine zu ihrer *Erlernung*. Wir werden uns daher im *Unterabschnitt 2.2* mit einigen Arbeiten zur maschinellen Erlernung anderer sprachlicher Regeln befassen, in der Hoffnung, dadurch weitere Anregungen für die Ausstattung unseres Algorithmus mit Lernstrategien zu erhalten.

In *Abschnitt 3* werden wir detailliert die Problemstellung und die Hypothesen dieser Arbeit darlegen, die wir in Abschnitt 1.1 ja erst vorläufig und informell skizziert hatten.

Die Problemlösung in Form des Programmsystems PRISM, das derivationsmorphologische Regeln erlernt und anwendet, werden die Abschnitte 4 und 5 behandeln. *Abschnitt 4* gibt zunächst einen einleitenden Überblick über das Programmsystem, wobei seine Hauptfunktionen und seine Makrostruktur beschrieben werden. Die Mikrostruktur und insbesondere die entwickelten Lernstrategien werden ausführlich dann in *Abschnitt 5* referiert.

Zur Überprüfung der Hypothesen mußten verschiedene Evaluierungsläufe mit dem Programm durchgeführt werden. Das für diese Evaluierungsläufe erforderliche sprachliche Ausgangsmaterial wird in *Abschnitt 6* beschrieben.

In *Abschnitt 7* werden wir PRISM evaluieren, indem wir die aufgestellten Hypothesen unter anderem anhand der Evaluierungsläufe überprüfen und deren statistische Ergebnisse interpretieren.

Abschnitt 8 nennt die Anwendungsmöglichkeiten von PRISM.

Der *Schlußabschnitt* (9) behandelt schließlich die Weiterentwicklungsmöglichkeiten des Programms. Außerdem enthält er einige selbstkritische Gedanken zum vorliegenden Ansatz.

Der *Anhang* enthält eine Auflistung des PRISM-Quellprogramms (10.1) und 2 Beispiele für das Input-Output-Verhalten des Programms (10.2 u. 10.3).

Am Ende des Buches finden sich das *Literaturverzeichnis* sowie, um den Zugriff zu bestimmten Informationen zu erleichtern, ein *Sachregister*.

2 DER FORSCHUNGSSTAND

Sprachlernalgorithmen wurden bisher hauptsächlich für die *Syntax* natürlicher Sprachen entwickelt. Einen guten und ausführlichen Überblick sowie eine Bewertung zu diesen Arbeiten liefert Steven Pinker (1979). Weitere, teilweise veraltete, teilweise sehr knappe Überblicke geben A. W. Biermann/J. A. Feldman (1972), Larry H. Reeker (1976) und Paul R. Cohen/Edward A. Feigenbaum (1982: 494-511). Ziel dieser Arbeiten zum Syntaxerwerb war entweder die Modellierung des Syntaxerwerbs beim Kind oder die Entwicklung von Systemen zur maschinellen Erzeugung syntaktischer Beschreibungen. Laut Pinker (1979: 218f.) weisen all diese Systeme im Hinblick auf die Erreichung der genannten Ziele noch erhebliche Mängel auf. Auch lassen sich die in diesen Systemen entwickelten Verfahren kaum auf das Lernen aus anderen Objektbereichen, wie etwa der Morphologie übertragen, da - wie Pinker (1979: 278) auch näher ausführt - die Darstellungsstrukturen, in denen die erlernten Daten und Regeln niedergelegt werden, sowie die Lernstrategien von den Erfordernissen des jeweiligen Objektbereiches abhängen, aus dem gelernt werden soll. Wir wollen daher auf diese Arbeiten hier nicht näher eingehen.

Die Entwicklung von Lernalgorithmen für den Bereich der *Morphologie* stand bislang weitgehend im Hintergrund. Eckhard Wolf bestätigt dies generell für Substitutionsbeziehungen, zu denen morphologische Ableitungsbeziehungen eine Untergruppe bilden. *Substitutionsbeziehungen* sind dabei alle und nur diejenigen zweistelligen Relationen, die zwischen 2 objektsprachlichen Zeichenketten[1] bestehen gdw. sich die zweite Zeichenkette

1 Unter einer *Zeichenkette* verstehen wir in dieser Arbeit eine Kette von Graphen. Die Zeichenkette mit der Länge 0 bezeichnen wir als *Nullstring* und notieren sie mit ''.

mittels einer oder mehrerer regelhafter Ersetzungen von Zeichenketten aus der ersten Zeichenkette erzeugen läßt. Wolf führt in seiner 1977 publizierten Dissertation "Vom Buchstaben zum Laut. Maschinelle Erzeugung und Erprobung von Umsetzautomaten am Beispiel Schriftenglisch - Phonologisches Englisch" aus:

> *"Untersuchungen zur automatischen Erzeugung von Umsetzungsautomaten liegen bisher nicht vor.* Ein Umsetzungsautomat wird aufgefaßt als eine operationalisierte und maschinell realisierbare Beschreibung einfacher Übersetzungszusammenhänge, die dazu dient, den Elementen eines von zwei Systemen, zwischen denen ein Umsetzungszusammenhang besteht, ein (oder mehrere) Element(e) des zweiten Systems zuzuordnen, so daß die Zuordnungen dem Umsetzungszusammenhang entsprechen. 'Automatische Erzeugung' soll heißen, daß ein solcher Umsetzungsautomat aufgrund einer Operationalisierung und maschinellen Realisierung der Entdeckungsprozeduren, die einer Beschreibung vorausgehen, automatisch erstellt werden kann. (Hervorhebung des Verf.)" (Wolf 1977: 1)

Wolfs Umsetzungsautomat besteht, wie seine weiteren Ausführungen verdeutlichen, aus einer von einem Lernalgorithmus erzeugten Menge von Substitutionsregeln und einem vorgegebenen Algorithmus zur Anwendung dieser Regeln.

Unsere Recherchen zum Forschungsstand ergaben allerdings, daß Arbeiten einerseits zu *ähnlichen Problemstellungen* wie in der vorliegenden Arbeit, andererseits zu *Teilproblemen* bereits durchgeführt wurden.

Diese Arbeiten lassen sich einordnen in die 2 Themenkomplexe:

- Arbeiten zur maschinellen Erlernung oder Simulation derivativer Wortbildungsregeln und
- Lernalgorithmen für andere sprachliche Substitutionsbeziehungen.

Die beiden folgenden Abschnitte 2.1 und 2.2 werden beide Gruppen von Arbeiten eingehender behandeln. Unser Ziel wird es dabei neben der bloßen Darstellung der Arbeiten sein, Schwachstellen und besondere Merkmale aufzuzeigen, die bei der Konstruktion unseres Systems besondere Beachtung verdienen müssen, sowie bereits wichtige Unterschiede zu unserem System herauszuarbeiten.

2.1 Arbeiten zur maschinellen Erlernung oder Simulation derivativer Wortbildungsregeln

In diesem Abschnitt werden wir eine Auswahl von Arbeiten zur maschinellen Erlernung oder Simulation derivativer Wortbildungsregeln vorstellen. Existierten mehrere Arbeiten mit (fast) identischer Problemstellung und Methodik, so haben wir uns darauf beschränkt, hier nur eine dieser Arbeiten stellvertretend für die anderen darzustellen. So haben wir z. B. auf eine Referierung der Arbeiten von M. Graitson (1975) und von Dieter Seelbach (1975) verzichtet und uns stellvertretend dafür auf das System von Milos G. Pacak und Arnold W. Pratt (vgl. Pratt/Pacak 1969 u. Pacak/Pratt 1978) beschränkt.

In den meisten hier referierten Arbeiten geht es ausschließlich um die maschinelle *Simulation*, bzw. Anwendung vorgegebener derivativer Wortbildungsregeln. Der interessantere Teil, die maschinelle *Erlernung* oder Entdeckung solcher Regeln ist Gegenstand nur einer Arbeit, nämlich der von Dieter Krallmann, Ute Krumnack und Hans Georg Soeffner (Krumnack 1972), und auch dort nur recht rudimentär. Weitere Arbeiten zu diesem zweiten Teilthema konnten wir bei unseren umfangreichen Recherchen zum Forschungsstand nicht finden. Die Entwicklung von Algorithmen zur maschinellen Erlernung derivativer Wortbildungsregeln scheint also ein bisher weitgehend vernachlässigtes Problem zu sein.

Die beiden Arbeiten, mit denen wir uns zunächst beschäftigen werden, also die Projekte von Karl-Dieter Bünting (1966 u. 1969) und von Krallmann, Krumnack und Soeffner, sind eher linguistisch-theoretischer Natur. In den 3 anschließend behandelten Arbeiten steht die Simulation morphologischer Ableitungsregeln dagegen in einem konkreten Anwendungsbezug: Sie soll innerhalb linguistisch orientierter Informationssysteme zur Erhöhung des Recall beim Retrieval dienen. Es handelt sich dabei um Komponenten der Systeme ISLIB (vgl. Bungarten 1978) und JUDO (vgl. Viets 1981) sowie um die Arbeiten von Milos G. Pacak und Arnold W. Pratt.

2.1.1 Bünting

Die frühesten Arbeiten zur maschinellen Simulation derivativer Wortbildungsregeln des Deutschen führte Karl-Dieter Bünting in der zweiten Hälfte der sechziger Jahre am Institut für Phonetik und Kommunikationsforschung (= IPK) der Universität Bonn durch. Über diese Arbeiten hat er u. a. in den folgenden Publikationen berichtet, auf die wir bei der folgenden Darstellung zurückgreifen:

Bünting, K.-D. (1966): Zur Erzeugung deutscher Wörter mit einem Computer. In: IPK-Forschungsberichte 5. Bonn.

Bünting, K.-D. (1969): Morphologische Strukturen deutscher Wörter: Ein Problem der linguistischen Datenverarbeitung. Bonn (Diss.).

Wir werden zunächst die in der ersten Publikation behandelte Arbeit wiedergeben: Bünting entwickelt und programmiert in dieser Arbeit einen Algorithmus, der derivative Wortbildungsregeln der deutschen Gegenwartssprache *simuliert*, jedoch - i. U. zu unserem System - *nicht erlernt*. Die Simulation dient dabei dem Zweck, eine vom Linguisten vorgegebene Beschreibung der Wortbildung auf Adäquatheit zu testen.

Bünting definiert zunächst eine Metasprache zur Beschreibung der Struktur abgeleiteter Wörter. Mit dieser Sprache - die wir hier nur informell darstellen - lassen sich abgeleitete Wörter beschreiben als eine Konkatenation von Morphemen, die in 3 Typen klassifiziert werden: *Kerne*, *Satelliten* und *Flexionsendungen*.

Kerne "sind die Morpheme, die in Verbindung mit einer Flexionsendung selbständig vorkommen können." (Bünting 1966: 6) - In herkömmlicher Terminologie handelt es sich bei den Kernen um Morpheme mit lexikalischer[2] Bedeutung, während Präfixe und Wortbildungssuffixe als Satelliten bezeichnet werden. - In Abhängigkeit davon, welcher Wortart ein Wort angehört, das aus der Konkatenation eines Kerns mit einer Flexionsendung besteht, lassen sich die Kerne in 3 Klassen einteilen:

2 'Lexikalisch' wird hier im Gegensatz zu 'grammatisch' verwendet.

- *Substantivkerne* (notiert: KS).
  Beispiel: {Mensch}, da 'Mensch' konkateniert mit dem Nullallomorph des Morphems {NOMINATIV SINGULAR} ein Substantiv ist.
- *Verbalkerne* (notiert: KV).
  Beispiel: {lieb}, da 'lieb' konkateniert mit dem Allomorph 'en' des Morphems {INFINITIV} ein Verbum ist.
- *Adjektivkerne* (notiert: KA).
  Beispiel: {lieb}, da 'lieb' konkateniert mit dem Nullallomorph des Morphems {POSITIV} ein Adjektiv (in der prädikativen Form im Positiv) ist.

Die beiden letzten Beispiele zeigen, daß ein Kern mehreren Kernklassen angehören kann.

Die Satelliten teilt Bünting in 18 Klassen ein, die durch zwei 3-wertige und eine 2-wertige Kategorie definiert sind:

- "Die Anfügkategorie: ein Satellit kann an Kerne einer oder mehrerer Klassen angefügt werden (notiert S, V, A)." (Bünting 1966: 6)
- "Die Resultatkategorie: Das Resultat der Zusammenfügung gehört in Verbindung mit einer Flexionsendung einer bestimmten Wortklasse an (notiert S, V, A)." (Bünting 1966: 6)
- "Die Stellungskategorie: die Satelliten unterscheiden sich nach ihrer Stellung relativ zum Kern in 'Präfixe' (*links* vom Kern) und 'Suffixe' (*rechts* vom Kern) (notiert L und R)." (Bünting 1966: 6)

Jede Satellitenklasse wird mit einem komplexen Symbol 'Wert der Anfügkategorie/Wert der Resultatkategorie/Wert der Stellungskategorie' bezeichnet. 'ung' ist demnach ein Satellit der Klasse V/S/R, da sich, wenn es mit dem *Verbal*kern 'leit' *rechts* konkateniert wird, das *Substantiv* 'Leitung' ergibt.

Wie sich die Wortbildungsstruktur deutscher abgeleiteter Wörter mit der dargestellten Metasprache beschreiben läßt, wollen wir uns kurz an einem Beispiel verdeutlichen: Gegeben sei die folgende Menge abgeleiteter Wörter {'unschön', 'unklug', 'vermenschlichen', 'lesbar', 'strafbar'}. Dann ist die folgende Strukturbeschreibung eine adäquate Beschreibung der Wortbildungsstruktur dieser Wortmenge, da sie die Erzeugung aller und nur der Wörter dieser Menge erlaubt:

Strukturtypen:[3]

```
A/A/L || KA,
A/V/L || KS || S/A/R || FV[4],
KV    || V/A/R.
```

Kern- und Satellitenklassen:

```
KA    = {'schön', 'klug'},
KS    = {'mensch'},
KV    = {'les', 'straf'},
A/A/L = {'un'},
A/V/L = {'ver'},
S/A/R = {'lich'},
V/A/R = {'bar'},
FV    = {'en'}.
```

Büntings *übergeordnetes Ziel* ist es, in der Metasprache die Wortbildungsstruktur der abgeleiteten Wörter der deutschen Gegenwartssprache zu beschreiben und die Beschreibung computativ auf ihre Adäquatheit zu überprüfen, indem sie einem Programm eingegeben wird und das Programm alle und nur die nach der Beschreibung zulässigen Wörter erzeugt. Die Beurteilung der Grammatikalität dieser Wörter durch den Linguisten oder einen geeigneten Sprachteilhaber ist dann gleichzeitig eine Beurteilung der Adäquatheit der Strukturbeschreibung. In der hier vorgestellten Arbeit wollte Bünting jedoch lediglich die Machbarkeit dieses Vorhabens experimentell überprüfen. Dafür hat er zu einer Teilmenge der abgeleiteten Wörter in einer intellektuellen Analyse eine Strukturbeschreibung erstellt. Diese Strukturbeschreibung bestand aus 8 Strukturtypen sowie den 3 Kern-, 9 Satelliten- und einer Flexionsendungsklasse mit insgesamt 17 Kernen, 12 Satelliten und einer Flexionsendung. Die Strukturbeschreibung wurde nun zur Erzeugung aller und nur der nach der Strukturbeschreibung zulässigen Wörter an ein Programm übergeben. Das Programm arbeitete sequentiell die vorgegebenen Strukturbeschreibungen ab.

> "Ein klassengleicher Kern nach dem anderen wird mit allen möglichen jeweils von der SB (= Strukturbeschreibung; Ergänzung des Verf.) geforderten Satelliten zusammengefügt." (Bünting 1966: 10)

3 Das Symbol || steht für die Konkatenation.
4 FV steht hier für 'verbale Flexionsendung'.

Bei der experimentellen Überprüfung der Machbarkeit seines Vorhabens hat Bünting folgende Mängel festgestellt:

- Einige Suffigierungen bewirken eine Umlautung des betonten Vokals des Kerns, mit dem der Suffixsatellit rechts konkateniert wird (Beispiel: 'täglich'). Die dargestellte Metasprache ermöglicht jedoch nicht die Beschreibung solcher Umlautungen.
- Manche Strukturbeschreibungen in der dargestellten Metasprache erlauben nicht nur die Erzeugung derjenigen abgeleiteten Wörter, durch deren (intellektuelle) Analyse sie gewonnen wurden, sondern auch zusätzlicher abgeleiteter Wörter. Diese zusätzlichen Wörter sind teilweise grammatisch zulässig. Jedoch werden auch viele der erzeugten Wörter "vom eigenen Sprachgefühl nicht akzeptiert werden." (Bünting 1966: 9) Die Strukturbeschreibungen enthalten also einerseits nützliche Generalisierungen, andererseits aber auch unerwünschte Übergeneralisierungen. Dieser Mangel dürfte zum Teil dadurch bedingt sein, daß die Metasprache keine Regeln enthält, die die Konkatenation bestimmter Morpheme explizit verbieten.

Auch bei Büntings Dissertation, "Morphologische Strukturen deutscher Wörter", handelt es sich - soweit sie hier von Interesse ist - allein um die maschinelle *Simulation* derivativer Wortbildungsregeln und *nicht* um ihre *Erlernung*. Die Vorgehensweise bei der Simulation derivativer Wortbildungsregeln ist der in der bereits dargestellten Arbeit von Bünting sehr ähnlich, wenngleich andere übergeordnete Ziele vorliegen: Es werden wiederum Kerne mit Satelliten und Flexionsmorphemen zu abgeleiteten Wörtern konkateniert. Bei dieser Konkatenation werden jedoch i. U. zu Büntings oben dargestellter Arbeit weder Subklassifizierungen der Kerne in Substantiv-, Verbal- und Adjektivkerne noch Beschränkungen der Satelliten und der Flexionsmorpheme hinsichtlich ihrer Konkatenierbarkeit mit bestimmten Kernklassen berücksichtigt, sondern es wird grundsätzlich *jeder* der vorgegebenen Kerne mit *jedem* der vorgegebenen Satelliten und Flexionsmorpheme konkateniert und lediglich der Wert der Stellungskategorie von Satelliten und Flexionsmorphemen bei ihrer Konkatenation mit einem Kern beachtet. Insgesamt werden 7 verschiedene Ableitungs-

typen untersucht. Bünting läßt dazu jeden von 2.759 Kernen[5], die er zuvor auf computergestütztem Wege aus einer maschinenlesbaren Kopie von Kandlers "Wortanalytischem Wörterbuch des Deutschen"[6] und durch eigene Untersuchungen gewonnen hat, vom Rechner mit jeder der folgenden Kombinationen aus Satelliten und Flexionsmorphemen konkatenieren (Die Konkatenation eines Kerns mit einer solchen Kombination besteht in der Ersetzung von '...' durch den Kern):[7]

```
      ... 'chen'  Beispiel: 'Brötchen',
      ... 'ung'   Beispiel: 'Bindung',
'be'  ... 'en'    Beispiel: 'beraten',
'be'  ... 'igen'  Beispiel: 'bereinigen',
      ... 'bar'   Beispiel: 'fruchtbar',
      ... 'lich'  Beispiel: 'glücklich',
'un'  ... 'lich'  Beispiel: 'unglücklich'.
```

Büntings übergeordnete Ziele bei dieser Arbeit sind nun u. a. folgende:

- Am Beispiel einer Untersuchung zur Wortbildungsmorphologie soll gezeigt werden, wie ein Computer bei sprachwissenschaftlichen Untersuchungen eingesetzt werden kann. - Diese Zielsetzung läßt sich nur aus der wissenschaftsgeschichtlichen Situation verstehen, in der Bünting die dargestellte Arbeit durchführte: In den sechziger Jahren waren computative Untersuchungen in der bundesrepublikanischen Sprachwissenschaft noch eine Seltenheit.
- Die künstlichen gebildeten Ableitungen sollen auf ihr tatsäch-

5 Dieses Korpus von Kernen betrachtet Bünting als repräsentativ für das Gesamtkorpus der gegenwartsdeutschen Kerne, "denn der gesamte Morphembestand einer Sprache wird, wenn überhaupt Schätzungen gewagt werden, auf ca. 4.000 bis 6.000 angesetzt" (Bünting 1969: 52).

6 Das "Wortanalytische Wörterbuch des Deutschen" wurde von G. Kandler in den sechziger Jahren am Sprachwissenschaftlichen Institut der Universität Bonn erstellt. Es enthält in maschinenlesbarer Form alle 117.370 Einträge der 3. Auflage des "Deutschen Wörterbuches" von Lutz Mackensen (1955). I. U. zum Mackensen liegen die Einträge im Wortanalytischen Wörterbuch jedoch segmentiert in Morpheme vor. Eine maschinenlesbare Kopie des Wortanalytischen Wörterbuchs befindet sich z. Z. in der Abteilung Linguistische Datenverarbeitung des Instituts für Kommunikationsforschung und Phonetik/Universität Bonn.

7 Vgl. Bünting 1969: 80.

liches Vorkommen in der deutschen Gegenwartssprache untersucht werden.

Zur Erreichung des letzten Ziels wurde maschinell überprüft, wie viele der künstlich erzeugten Ableitungen im "Wortanalytischen Wörterbuch des Deutschen" belegt sind, und es wurde der prozentuale Anteil der *belegten* künstlichen erzeugten Ableitungen an der *Gesamtzahl* künstlich erzeugter Ableitungen berechnet. Für die 7 betrachteten Ableitungstypen ergab sich jeweils folgender prozentualer Anteil (hier abgekürzt mit PA):

Ableitungstyp	PA
... 'chen'	3,2%
... 'ung'	11,3%
'be' ... 'en'	12,6%
'be' ... 'igen'	0,7%
... 'bar'	3,2%
... 'lich'	5,0%
'un' ... 'lich'	1,1%

Abb. 2.1 (nach Bünting 1969: 82)

Ein Vergleich des in dieser Arbeit zu entwickelnden Systems mit Büntings Systemen im Hinblick auf die maschinelle *Simulation* derivativer Wortbildungsregeln zeigt, daß verschiedene Problemstellungen vorliegen: Unser System soll aus vorgegebenen Wortformen andere Wortformen erzeugen, die zu den vorgegebenen Wortformen in einer morphologischen Ableitungsbeziehung stehen. Dagegen sollen Büntings Systeme Kerne, Satelliten und Flexionsendungen zu derivierten Wörtern konkatenieren.

2.1.2 Krallmann/Krumnack/Soeffner

Der einzige bisher bekannte Versuch, einen Algorithmus zur Erlernung und Simulation derivativer Wortbildungsregeln zu entwickeln, wurde von Dieter Krallmann, Ute Krumnack und Hans Georg Soeffner unternommen. Sie arbeiteten Ende der sechziger, Anfang der siebziger Jahre am Institut für Kommunikationsforschung und Phonetik der Universität Bonn an einem Verfahren zur maschinellen Erzeugung derivativer Wortbildungsregeln. Über ihre Arbeiten wird u. a. in folgendem Aufsatz berichtet,

den wir auch unseren Ausführungen zugrunde legen:

> Krumnack, U. (1972): Kriterien zur Klassifizierung von Segmenten für die automatische Analyse und Synthese. In: IPK-Forschungsberichte. Bd. 43. Hamburg. S. 67-148.

Diejenigen Ziele des Vorhabens, die hier von Interesse sind, waren:

1. Die maschinelle Ermittlung von Morphemklassen und von Regeln, die die Kombinierbarkeit der Elemente dieser Morphemklassen zu abgeleiteten Wörtern festlegen.
2. Im Anschluß an 1 die maschinelle Anwendung der ermittelten Regeln zwecks Erzeugung abgeleiteter Wörter.

Während das erste Ziel weitgehend erreicht wurde, ist über die Arbeiten zur zweiten Problemstellung aus der Literatur nichts zu erfahren. Im folgenden werden wir zunächst referieren, wie die erste Problemstellung bearbeitet wurde. Anschließend wollen wir feststellen, ob es sich bei dem Verfahren von Krallmann, Krumnack und Soeffner tatsächlich um eine maschinelle *Erlernung* derivativer Wortbildungsregeln handelt, und es soll ein Vergleich gezogen werden zwischen ihrem Verfahren und dem Vorhaben der vorliegenden Arbeit.

Als Ausgangsmaterial für die maschinelle Ermittlung der Morphemklassen und Regeln diente das maschinenlesbar vorliegende "Wortanalytische Wörterbuch des Deutschen".[8] Aus diesem Gesamtmaterial wurde ein sogenannter 'Reduzierter Mackensen' extrahiert, der als Grundlage für die weiteren Arbeiten diente und der mit 68.508 Einträgen nur die (in Morpheme) *segmentierten deutschen* Wörter des Wortanalytischen Wörterbuchs mit ihren Wortartangaben enthält. Der Reduzierte Mackensen enthält also beispielsweise nicht:

- alle Fremdwörter,
- alle *nicht*-segmentierbaren deutschen Wörter.

Diese Reduzierung wurde vorgenommen, da die Untersuchung nur auf die deutsche Wortbildung gerichtet sein sollte und da, so Krumnack, nicht-segmentierbare Wörter des rein deutschen Wort-

8 Nähere Informationen zum Wortanalytischen Wörterbuch findet man in der obigen Darstellung der Arbeiten von Bünting in Abschnitt 2.1.1.

schatzes vor allem für die Morphemklassifizierung und die Regelermittlung, aber auch bei der intendierten späteren Regelanwendung unergiebig seien, "da sie vielfach nur einmal in der gesamten Wortliste vertreten sind." (Krumnack 1972: 71)

Die im Reduzierten Mackensen vorhandenen Morpheme, die im folgenden auch als Segmente bezeichnet werden, wurden maschinell in 3 Morphemklassen eingeteilt:

- *Basissegmente*: "Ein Basissegment ist ein Segment, das eine Basisposition einnehmen kann. Eine Basisposition ist in einsegmentigen Wörtern die einzig vorhandene Position; in mehrsegmentigen Wörtern eine Position in einer Kette von Positionen, die sich aufgrund der Belegung der anderen Positionen (...; Auslassung des Verf.) als Basisposition ausweist." (Krumnack 1972: 82)

- *Rechtssegmente*: "Rechtssegmente sind diejenigen Segmente, die rechts von einer Basis stehen, ohne selbst Basis zu sein." "Rechtssegmente können in einem Wort aneinandergereiht erscheinen, sie müssen aber als letzten linken Nachbarn eine Basis haben." (Krumnack 1972: 84)

- *Linkssegmente*: "Linkssegmente sind diejenigen Segmente, die links von einer Basis stehen, ohne selbst Basis zu sein." (Krumnack 1972: 85)

Linkssegmente können - ebenso wie die Rechtssegmente - in einem Wort aneinandergereiht erscheinen, sie müssen als letzten rechten Nachbarn eine Basis haben.

Im großen und ganzen bezeichnen die Termini Basissegment, Rechtssegment und Linkssegment dieselben Arten von Morphemen wie jeweils die traditionellen Termini lexikalisches Morphem, Wortbildungssuffix und Wortbildungspräfix.

Die *wichtigsten* Phasen bei der maschinellen Einteilung der Morpheme in die 3 Morphemklassen seien im folgenden wiedergegeben:

1. "Alle einsegmentigen Wörter werden per definitionem als Basissegmente angesehen" (Krumnack 1972: 88) und in einer ersten tentativen Liste von Basissegmenten zusammengefaßt. Dabei wird zu jedem Basissegment aus dem Reduzierten Mackensen die Wortartangabe des jeweiligen einsegmentigen Wortes übernommen.

"Die Segmente dieser Liste werden mit den zweisegmentigen Wörtern verglichen; dabei werden die Stellen ihres Vorkommens im Wort als Basispositionen markiert." (Krumnack 1972: 89)

2. "Die zweisegmentigen Wörter werden mithilfe der bereits markierten Segmente weiter analysiert, und zwar in zwei (...; Auslassung des Verf.) Schritten (...; Auslassung des Verf.).
   a) Bei den Wörtern, die nur ein Basissegment enthalten, werden die Segmente vor dieser Basis als Linkssegmente gelistet, die nach der Basis als Rechtssegmente. (...; Auslassung des Verf.)
   b) Die Wörter mit zwei Basismarkierungen bzw. mit mindestens einer markierten Basis in zweiter Position werden daraufhin untersucht, ob es sich um echte Zusammensetzungen handelt, oder ob das zweite Segment nicht vielmehr ein Homograph ist, der sowohl Basis- wie Rechtssegment sein kann. Als Entscheidungskriterium gilt die Phrasendeterminationsmarkierung (= Wortartangabe; Ergänzung des Verf.)." (Krumnack 1972: 90f.)

Es wird grundsätzlich angenommen, daß die Wortartzugehörigkeit von Komposita der deutschen Gegenwartssprache gleich der Wortartzugehörigkeit des Basissegments in letzter Position der Komposita ist:

"Daraus folgt, daß bei Wörtern, deren Phrasendetermination nicht mit der des Ausgangswortes des zweiten Segmentes übereinstimmt, diese Veränderung das Auftreten von Rechts- oder ggf. auch Linkssegmenten signalisiert. Diese Wörter werden daraufhin nicht weiter als Zusammensetzungen angesehen; das zweite Segment gilt im analysierten Wort als Rechtssegment und wird für die künftigen Arbeitsstufen als Homograph klassifiziert." (Krumnack 1972: 91)

3. "Die Listen der ermittelten Rechts- und Linkssegmente werden mit dem Gesamt-Mackensen verglichen; dabei soll festgestellt werden, welche Segmente dort noch als einsegmentige Wörter vorkommen (...; Auslassung des Verf.). Die gefundenen Segmente erhalten denselben Status wie die unter 1.) genannten; sie werden also bis zum Erweis des Gegenteils als Basissegmente angesehen (und nicht als Homographen), da sie nur zufälligerweise nicht in der ersten Liste der Basissegmente aufgenommen worden sind." (Krumnack 1972: 92)

4. "Mithilfe der Rechtssegmente werden die zweisegmentigen Wörter erneut analysiert, wobei alle Segmente, die vor einem Segment dieser Liste stehen, als Basissegmente klassifiziert (...; Auslassung des Verf.) werden. Ein entsprechendes Verfahren wird mit den Linkssegmenten durchgeführt". (Krumnack 1972: 92f.)

5. Die Listen der ermittelten Rechts- und Linkssegmente werden mit der Liste der ermittelten Basissegmente verglichen, um

Homographen festzustellen.

Bis hierhin haben Krallmann, Krumnack und Soeffner die Schritte zur Einteilung von Morphemen in die 3 Morphemklassen Linkssegmente, Basissegmente und Rechtssegmente automatisiert. Für den anschließenden Übergang zur Klassifizierung von Morphemen aus Wörtern mit *mehr als 2 Morphemen* wurden die Verfahrensschritte nicht erarbeitet.

Auch die Kombinierbarkeit der Elemente der 3 Morphemklassen zu abgeleiteten und zusammengesetzten Wörtern wurde zunächst nur für zweisegmentige Wörter untersucht. Dabei wurde festgestellt,

> "wie groß überhaupt bei den bislang klassifizierten Basissegmenten die Neigung ist, sich mit Positionssegmenten (= Links- und Rechtssegmente; Ergänzung des Verf.) zu verbinden, welche verschiedenen Positionssegmente als Vorgänger bzw. Nachfolger für die einzelnen Basen belegt sind und ob sich im Hinblick auf die Kombination mit gleichen Positionssegmenten bestimmte Gruppen von Basissegmenten zusammenstellen lassen." (Krumnack 1972: 121f.)

Die Zusammenstellung der möglichen Kombinationen läßt sich verwenden, "um daraus das Material für Algorithmen zur stufenweisen Synthese komplexer Präfix- oder Suffixbildungen zu gewinnen." (Krumnack 1972: 119) Auf das verwendete maschinelle Verfahren zur Zusammenstellung der Morphemkombinationen soll hier nicht eingegangen werden, da es relativ trivial ist.

Abschließend sei folgendes zur Arbeit von Krallmann, Krumnack und Soeffner festgehalten:

- Sie haben bei ihrem Verfahren zwar die Entwicklung einer Art Lernalgorithmus intendiert, jedoch ist dieses Ziel nicht erreicht worden: Das entwickelte Verfahren leistet nicht die maschinelle Erzeugung maschinell ausführbarer Wortbildungsregeln oder eines maschinell ausführbaren Algorithmus.
- Eine linguistische Inadäquatheit des Verfahrens von Krallmann, Krumnack und Soeffner, die diese auch gesehen haben, ist das Fehlen von Regeln zur Beschreibung von Stammveränderungen (wie z. B. Umlaut und Ablaut), die bei derivativen Wortbildungen auftreten.
- Ihre Zielsetzung unterscheidet sich etwas von der der vorliegenden Arbeit: Ihr Verfahren soll Regeln ermitteln, mit-

tels derer sich Morpheme zu abgeleiteten Wörtern konkatenieren lassen. Es geht also um die maschinelle Konstruktion einer *Synthese*grammatik. Demgegenüber soll von dem in der vorliegenden Arbeit zu entwickelnden System eine *Transformations*grammatik maschinell erzeugt werden: Es sollen Regeln erlernt werden, mittels derer sich aus Wortformen andere - zu ihnen in einer morphologischen Ableitungsbeziehung stehende - Wortformen bilden lassen.

2.1.3 Pacak/Pratt

Milos G. Pacak und Arnold W. Pratt arbeiten an den National Institutes of Health/Bethesda (Maryland, U.S.A.) seit Ende der sechziger Jahre an einem maschinellen Verfahren zur Bestimmung der Wortartzugehörigkeit vorgegebener Wortformen und zur Transformation dieser Wortformen in andere, semantisch ähnliche Wortformen. Die dabei erzeugten Wortformen stehen zu den vorgegebenen Wortformen entweder in einer morphologischen Ableitungsbeziehung, oder sie besitzen lediglich eine andere flexionsmorphologische Merkmalsausprägung. Über ihr Verfahren haben Pacak und Pratt u. a. in den folgenden Publikationen berichtet, auf die wir uns bei unserer Darstellung auch beziehen:

Pratt, A. W./Pacak, M. G. (1969): Identification and transformation of terminal morphemes in medical English. In: Methodik der Information in der Medizin 2. S. 84-90.

Pacak, M. G./Pratt, A. W. (1978): Identification and transformation of terminal morphemes in medical English. Part 2. In: Methods of information in medicine 2. S. 95-100.

Das Verfahren kommt in einem maschinellen Informationssystem zur Verarbeitung englischer medizinischer Fachtexte zur Anwendung. In diesem Informationssystem wird zum maschinellen Indexing ein Thesaurus verwendet, der nicht alle flexionsmorphologischen Varianten der potentiellen Deskriptoren enthält und auch nicht alle zu den Deskriptoren in einer morphologischen Ableitungsbeziehung stehenden, ihnen semantisch ähnlichen Wortformen. Um nun einem maschinell zu indexierenden Fachtext auch dann einen Deskriptor zuweisen zu können, wenn er nicht die im Thesaurus verzeichnete Form dieses Deskriptors enthält, son-

dern nur morphologische Ableitungen oder flexionsmorphologische Varianten dieses Deskriptors, haben Pacak und Pratt ein Transformationsverfahren entwickelt, das die maschinelle Überführung von Wortformen in andere Wortformen leistet, welche zu ihnen entweder in einer morphologischen Ableitungsbeziehung stehen oder auch nur eine andere flexionsmorphologische Merkmalsausprägung besitzen. Im einzelnen werden folgende Transformationen durchgeführt:

- Adjektive und Partizipien werden transformiert in Substantive.
  Beispiele: 'epiglottic' <Kehldeckel- (ADJ)> wird transformiert in 'epiglottis' <Kehldeckel (N)>, 'scarifying' <skarifizierend> wird zu 'scarification' <Skarifizierung>.
- Substantive werden transformiert in Substantive.
  Beispiel: 'dysentery' <Ruhr> wird zu 'dysenteria' <Ruhr>.
- Substantive werden transformiert in Adjektive.
  Beispiel: 'morphology' <Morphologie> wird zu 'morphologic' <morphologisch>.
- Substantive im Plural werden zu Substantiven im Singular.
  Beispiel: 'appendices' <Blinddärme> wird zu 'appendix' <Blinddarm>.

Die für die Transformation erforderlichen Regeln wurden intellektuell und nicht computativ erstellt. Sie nehmen ausschließlich auf die graphische Gestalt der zu transformierenden Wortformen Bezug. Außer zur Transformation dienen die Regeln noch zur Erkennung der Wortartzugehörigkeit der zu transformierenden und der transformierten Wortform. Sie lassen sich u. a. in der folgenden allgemeinen Form darstellen:[9]

$$X(WA) \rightarrow a_1Y_1(WA_1)|a_2Y_2(WA_2)| \ \ldots \ |a_{n-1}Y_{n-1}(WA_{n-1})|a_nY_n(WA_n)$$

Diese allgemeine Regelform ist folgendermaßen zu lesen: Endet eine zu transformierende Wortform A auf die Zeichenkette X, so gehört A der Wortart WA an und die letzten a_1, a_2, ..., a_{n-1}

9 Vgl. dazu Pratt/Pacak 1969: 89 oder Pacak/Pratt 1978: 98. Pacak und Pratt notieren ihre Regeln etwas anders, als hier angegeben. Da ihre Notation jedoch recht ungewöhnlich ist, wird hier zum leichteren Verständnis eine anschaulichere Notation verwendet, die der von Pratt und Pacak natürlich äquivalent ist.

oder a_n Zeichen von A sind durch die Zeichenkette Y_1, Y_2, ..., Y_{n-1} oder Y_n respektiv zu ersetzen, um eine potentielle Wortform B mit der Wortartzugehörigkeit WA_1, WA_2, ..., WA_{n-1} oder WA_n zu erhalten, die zu A entweder in einer morphologischen Ableitungsbeziehung steht oder lediglich eine andere flexionsmorphologische Merkmalsausprägung hat als A.

Von den n erzeugten potentiellen Wortformen ist nur eine die gewünschte Wortform. Welches die gewünschte Wortform ist, wird festgestellt durch Vergleich jeder erzeugten Wortform mit den Thesauruseinträgen. Die erste erzeugte Wortform, die auch im Thesaurus verzeichnet ist, gilt als die zulässige Wortform. Beispiel:[10] Gegeben sei die Regel

'ic'(ADJ) → 2''(N)|2'a'(N)|2'e'(N)|2'y'(N)|2'is'(N)|2'u'(N)|
2'um'(N)|2'on'(N)

und die zu transformierende Wortform

'epiglottic'.

Durch Anwendung der Regel auf 'epiglottic' werden nacheinander die Formen 'epiglott', 'epiglotta', 'epiglotte' und 'epiglotty' erzeugt, vergeblich im Thesaurus aufgesucht und somit als unzulässige Formen eingestuft. Die daraufhin erzeugte Form 'epiglottis' wird im Thesaurus gefunden und daher als zulässige Form angenommen. Die weiteren Transformationen, bei den 'epiglottu', 'epiglottum' und 'epiglotton' erzeugt würde, werden nicht mehr angewendet.

Die Reihenfolge, in der die Regeln angewendet werden dürfen, ist festgelegt. Es handelt sich also um geordnete Regelmengen. Und zwar werden die Regeln so geordnet, daß Regeln mit längeren zu ersetzenden Zeichenketten stets *vor* solchen mit kürzeren zu ersetzenden Zeichenketten anzuwenden versucht werden. Somit kann jede Regel potentiell zur Anwendung kommen.

Eine *Erlernung* derivativer Wortbildungsregeln leistet das Verfahren von Pacak und Pratt nicht. *Hinsichtlich der Simulation derivativer Wortbildungsregeln* kommt es von allen uns bekannten Verfahren der Problemstellung unseres Vorhabens am nächsten: Bei

10 Nach Pratt/Pacak 1969: 86f.

den übrigen in Abschnitt 2.1 dargestellten Arbeiten geht es entweder darum, Morpheme zu abgeleiteten Wörtern zu konkatenieren (vgl. Bünting und Krallmann/Krumnack/Soeffner) oder darum, morphologische Ableitungsregeln durch eine Lexikonaufsuche zu simulieren (vgl. JUDO). Dagegen versuchen Pacak und Pratt, zu vorgegebenen Wortformen mittels Transformationen andere Wortformen zu erzeugen, die zu den vorgegebenen Wortformen in einer morphologischen Ableitungsbeziehung stehen. Wie sich bei der Beschreibung unseres Systems in den Abschnitten 3-5 zeigen wird, bestehen bei der Simulation derivativer Wortbildungsregeln nicht nur in der Problemstellung, sondern auch in der Methodik einige auffallende Parallelen zwischen unserem System und dem von Pacak und Pratt. Um diese Parallelen und auch die Unterschiede später leichter erkennbar werden zu lassen, seien hier abschließend einige Eigenschaften des Systems von Pacak und Pratt noch einmal besonders hervorgehoben:

- Die abgeleiteten Wortformen werden aus den vorgegebenen Wortformen grundsätzlich durch Ersetzung eines rechten Teilstrings[11] erzeugt.
- Die Regeln erlauben die Ersetzung eines rechten Teilstrings einer vorgegebenen Wortform durch mehrere verschiedene Zeichenketten. Sie können also auch einmehrdeutige Transformationen leisten, wobei allerdings nur eines der Transformationsergebnisse eine zulässige Wortform ist. Welche der er-

11 Als *Teilstring* einer Zeichenkette Z bezeichnen wir jeden kontinuierlichen (d. h. nicht unterbrochenen) Ausschnitt aus Z, dessen Länge *größer oder gleich 0 und kleiner oder gleich der Länge von Z* ist. Demnach ist der *Nullstring*, also die Zeichenkette mit der Länge 0, Teilstring jeder Zeichenkette. (Wir notieren den Nullstring mit ''). Als *rechten Teilstring* einer Zeichenkette Z bezeichnen wir jeden Teilstring, auf den Z endet. Analog dazu bezeichnen wir als *linken Teilstring* einer Zeichenkette Z jeden Teilstring, mit dem Z beginnt. Beispiele: Die Menge aller möglichen Teilstrings zur Zeichenkette 'es' ist {'es', 'e', 's', ''}. Menge aller möglichen rechten Teilstrings zu 'es' ist {'es', 's', ''}. Menge aller möglichen linken Teilstrings ist {'es', 'e', ''}.
Als besondere Untergruppe der Teilstrings führen wir die *echten Teilstrings* ein. Sie müssen stets kürzer sein als Z. Demnach ist z. B. 'e' echter (linker) Teilstring von 'es'. 'es' ist dagegen nicht echter Teilstring von 'es'.

erzeugten Zeichenketten die zulässige Wortform ist, wird durch eine Lexikonaufsuche geklärt.

- Die von Pacak und Pratt verwendeten Regeln sind kontextsensitiv. Da die Kontextsensitivität vermutlich nicht ohne weiteres einzusehen ist, soll sie hier mit einem Beispiel verdeutlicht werden: Die kontextsensitive Regel

  'Ersetze 'cy' am Wortende durch 't', falls es unmittelbar rechts von 'an' steht.'

  wird mit der Regelform von Pacak und Pratt folgendermaßen ausgedrückt:

  'ancy'(N) → 2't'(ADJ)

  Durch diese Regel werden aus Nomina Adjektive erzeugt. Aus 'pregnancy' <Schwangerschaft> würde mit dieser Regel also 'pregnant' <schwanger> erzeugt.
- Die Regeln sind geordnet, d. h., sie dürfen nur in einer bestimmten Reihenfolge anzuwenden versucht werden.
- Pacak und Pratt verwenden neben dem bisher dargestellten Regeltyp noch sogenannte exclusion transformation rules, die sie bei der Transformation von Nomina in Adjektive einsetzen:

  "The purpose of so-called exlusion (*sic*) transformation rules is to eliminate certain noun-to-adjective transforms which are not admissible because of the absence of a corresponding equivalent of adjectival form. For example, the nominal word forms with the suffix 'itis' (e. g., 'appendicitis' (<Blinddarmentzündung>; Ergänzung des Verf.)), 'ectomy' (e. g., 'appendectomy' (<Blinddarmoperation>; Ergänzung des Verf.)), and others should be excluded from transformations before trying to apply the transformation rules for the suffixes 'is' or 'y'." (Pacak/Pratt 1978: 99)

Ein ähnliches Verfahren wie Pacak und Pratt hat M. Graitson - ebenfalls an den National Institutes of Health/Bethesda (Maryland, U.S.A.) - in der ersten Hälfte der siebziger Jahre für *französische* medizinische Fachtexte entwickelt. Auf sein Verfahren soll hier nicht näher eingegangen werden. Es unterscheidet sich methodisch nämlich nur unwesentlich von dem oben beschriebenen Verfahren von Pacak und Pratt für *englische* medizinische Fachtexte. Detaillierte Informationen zu Graitsons Verfahren findet man in dem Aufsatz:

Graitson, M. (1975): Identification et transformation automatique des morphèmes terminaux dans le lexique médical français. In: Cahiers de lexicologie. Revue internationale de lexicologie et de lexicographie 1. S. 85-109.

Ein weiteres Konzept, das dem von Pacak und Pratt methodisch recht ähnlich ist, hat Dieter Seelbach für die Nominalisierung von Partizipien und Adjektiven in englischen Abstracts von Fachtexten zur Ernährungswissenschaft vorgestellt. Anwendungsrahmen für sein Verfahren ist ebenfalls das maschinelle Indexing. Da Seelbachs Verfahren keine wesentlichen methodischen Neuerungen zu dem von Pacak und Pratt aufweist, die auch für die vorliegende Arbeit von Interesse wären, können wir von näheren Ausführungen dazu absehen. Eine Darstellung von Seelbachs Vorschlägen findet man in dem Buch:

Seelbach, D. (1975): Computerlinguistik und Dokumentation. München. S. 62-74.

2.1.4 ISLIB

Im Zusammenhang mit der Entwicklung des natürlichsprachlich fundierten Informations- und Problemlösungssystems ISLIB[12] (= *I*nformations*s*ystem auf *l*inguistischer *B*asis), das ab 1973 am Institut für deutsche Sprache/Mannheim entwickelt wurde, hat Theo Bungarten versucht, einen Algorithmus zu entwerfen, der Verben der deutschen Gegenwartssprache mit ihren Aktanten und Circonstanten[13] in deverbative Nominalgruppen transformiert. Der Algorithmus soll also beispielsweise die diskontinuierliche Verbform 'lagert ... ab' mit ihren Aktanten 'viel Kalk' und 'sich' sowie dem Circonstanten 'bei der Verdunstung' in dem

12 Einführende Informationen zur Konzeption von ISLIB findet man bei Ungeheuer (1977).

13 Bungarten verwendet anstelle der aus Tesnières Dependenzgrammatik stammenden Termini 'Aktant' und 'Circonstant' den ebenfalls aus der Dependenzgrammatik stammenden, aber in diesem Zusammenhang nicht angebrachten Terminus 'Valenz'. (Eine gute Darstellung der Theorie der Dependenzgrammatik mit einer Erläuterung der erwähnten Termini findet man bei Tesnière (1953)).

Satz

'Viel Kalk lagert sich bei der Verdunstung ab.'

überführen in die deverbative Nominalgruppe

'die Ablagerung von viel Kalk bei der Verdunstung'.

Die Funktion dieses Algorithmus innerhalb des Systems ISLIB sollte darin bestehen, die Berücksichtigung von Synonymiebeziehungen zwischen Verben, ihren Aktanten und Circonstanten einerseits und deverbativen Nominalgruppen andererseits zu ermöglichen, um dadurch den Recall beim Retrieval zu erhöhen. (Den Zusammenhang zwischen Erhöhung des Recall und der Berücksichtigung solcher Synonymiebeziehungen werden wir in Abschnitt 2.1.5 noch ausführlicher erläutern.)

Die folgenden, eingehenderen Ausführungen zu Bungartens Arbeit basieren auf dem Aufsatz

> Bungarten, Th. (1978): Die Überführung ausgewählter Verben und ihrer Valenzen zu Deverbativgruppen in der Nominaltransformation (konzipiert für einen linguistischen Transformationsalgorithmus in einem deutschsprachigen Informations- und Problemlösungssystem (ISLIB)). In: Grazer linguistische Studien 8. S. 5-30.

Bei der Darstellung der Transformationsregeln wollen wir uns auf *die* Regeln beschränken, die im Rahmen der vorliegenden Arbeit vornehmlich von Interesse sind, nämlich die Regeln zur Simulation derivativer Wortbildungsregeln, die bei Bungarten ausschließlich zur Bildung von Nomina actionis zu Verben dienen. Dagegen wollen wir diejenigen Regeln vernachlässigen, die - gleichzeitig zur Bildung des Nomen actionis - die Aktanten und Circonstanten des Verbs in Attribute des Nomen actionis überführen.

Bungarten teilt zunächst die Verben in 4 Klassen ein:

> "Die Verben mit ihren Valenzrahmen verteilen sich entsprechend der möglichen Anzahl und Qualität ihrer Leerstellen auf die Satzrelationen *Handlung*, *Vorgang*, *Merkmalzuordnung*, und *Zustand* (...; Auslassung des Verf.). Die Zuordnung eines Verbs zu einer oder mehreren dieser vier sogenannten Satzrelationen erfolgt aufgrund bestimmter syntaktischer und semantischer Kriterien, deren methodologisch-theoretische Angemessenheit wie auch die der Satzrelationen selbst hier nicht weiter diskutiert werden sollen." (Bungarten 1978: 12)

Wie die folgende, exemplarische Auflistung einiger Verben zu jeder dieser Klassen verdeutlicht, können Verben durchaus mehreren Klassen angehören. Die Zugehörigkeit eines Verbs zu mehreren Klassen gilt jedoch meist nur, solange das Verb in Isolation betrachtet wird. In Vertextung ist die Klassenzugehörigkeit eines solchen Verbs aufgrund des semantisch-syntaktischen Kontextes in der Regel eindeutig. Um dies zu verdeutlichen, führt Bungarten (1978: 10-13) zu den Verben jeweils eine vereindeutigende Vertextung an:

Satzrelation Handlung

Verb	Vereindeutigende Vertextung
'abfallen'	'Er fiel vom Glauben ab.'
'abschließen'	'Er schließt die Tür mit dem Schlüssel ab.'
'bedrohen'	'Der Mörder bedroht das Opfer mit dem Messer.'
'drohen'	'Der Mörder droht seinem Opfer mit dem Messer.'
'transportieren'	'Die Firma transportiert den Müll in geschlossenen Behältern auf die Müllkippe.'

Abb. 2.2

Satzrelation Vorgang

Verb	Vereindeutigende Vertextung
'abfallen'	'Die Blätter fallen von den Zweigen (durch den Wind) ab.'
'bedrohen'	'Die Flut bedroht die Deiche.'
'lösen'	'Der Aufkleber löst sich (durch Feuchtigkeit) von der Unterlage.'
'transportieren'	'Lkws transportieren den Müll auf die Müllkippe.'

Abb. 2.3

Satzrelation Merkmalzuordnung

Verb	Vereindeutigende Vertextung
'abfallen'	'Asche fällt bei jeder Verbrennung ab.'
'abnehmen'	'Ihre Auffassungsgabe nimmt ab.'
'ansteigen'	'Der Berg steigt nach Westen hin an.'
'steigen'	'Die Straße steigt zum Berg hin.'

Abb. 2.4

Satzrelation Zustand

Verb	Vereindeutigende Vertextung
'abfallen'	'Die Blätter sind von den Zweigen (durch den Wind) abgefallen.'
'abschließen'	'Die Bilanz schließt mit Verlust ab.'
'bedrohen'	'Das Gesetz bedroht unerlaubtes Deponieren von Müll mit Haft.'
'lösen'	'Das Blatt hat sich (durch Witterungseinflüsse) aus dem Buch gelöst.'

Abb. 2.5

Die Art, wie zu einem Verb nun das entsprechende Nomen actionis gebildet wird, hängt nach Bungarten von der Klassenzugehörigkeit des Verbs ab:

- Bei den Verben der *Satzrelation Handlung* sind - abgesehen von wenigen Ausnahmen - Nominalisierungen in Form des substantivierten Infinitivs (Beispiel: 'abschließen' <==> 'Abschliessen')[14] und durch Konkatenation des Präsensstamms mit dem Suffix 'ung' (Beispiel: 'drohen' <==> 'Drohung') möglich (vgl. Bungarten 1978: 17).
- Bei den Verben der *Satzrelation Vorgang* ist Nominalisierung in Form des substantivierten Infinitivs "fast ausnahmslos möglich" (Bungarten 1978: 19).
- Die Verben der *Satzrelation Merkmalzuordnung* können auf die folgenden Arten nominalisiert werden (vgl. Bungarten 1978: 19):
  - Bildung des substantivierten Infinitivs.
    Beispiel: 'abfallen' <==> 'Abfallen'.
  - Konkatenation des Präsensstamms mit dem Suffix 'ung'.
    Beispiel: 'steigen' <==> 'Steigung'.
  - Bildung des Präsensstamms.
    Beispiel: 'abfallen' <==> 'Abfall'.
  - Konkatenation des Präteritalstamms mit 'e'.
    Beispiel: 'abnehmen' <==> 'Abnahme'.
- "Im Vergleich zu den übrigen Satzrelationen sind die Nominalisierungsmöglichkeiten von Verben der SRZ (= Satzrelation Zustand; Ergänzung des Verf.) quantitativ am eingeschränk-

14 Das Symbol <==> wird hier in der Bedeutung 'steht in einer morphologischen Ableitungsbeziehung zu' verwendet.

testen. Dennoch sind für die Verben, für die Transforme möglich sind, sehr unterschiedliche Formen festzustellen:" (Bungarten 1978: 19)

- Substantivierter Infinitiv.
  Beispiel: 'lösen' <==> 'Lösen'.
- Bildung des Präsensstamms.
  Beispiel: 'abfallen' <==> 'Abfall'.
- Konkatenation des Präsensstamms mit dem Suffix 'ung'.
  Beispiel: 'bedrohen' <==> 'Bedrohung'.
- Konkatenation des unflektierten Partizips Perfekt mit dem Suffix 'heit'.
  Beispiel: 'lösen' <==> 'Gelöstheit'.
- Sonderformen.
  Beispiel: 'abschließen' <==> 'Abschluß'.

Eine algorithmische Fassung seiner Nominalisierungsregeln gibt Bungarten nicht an. Bei einer Algorithmisierung des Verfahrens dürften auch erhebliche Schwierigkeiten auftreten. Eine dieser Schwierigkeiten besteht darin, zu einem vertexteten Verb, das mehreren Klassen angehören kann, die in der Vertextung aktualisierte Klassenzugehörigkeit zu ermitteln. Kritisch anzumerken ist auch, daß die Nominalisierung mittels Konkatenation des unflektierten Partizips Perfekt mit dem Suffix 'heit' (Beispiel: 'lösen' <==> 'Gelöstheit') gar nicht zu dem erforderlichen Nomen actionis führt, geschweige denn zu einer syntaktisch wohlgeformten und semantisch sinnvollen deverbativen Nominalgruppe (*'Die Gelöstheit des Blattes aus dem Buch (durch Witterungseinflüsse)').

Ein Vergleich von Bungartens Verfahren mit unserem System läßt sich schwer durchführen, da es sich bei Bungartens Verfahrens lediglich um einen Entwurf handelt. Als deutlicher Unterschied bleibt auf jeden Fall festzuhalten: Bungarten versucht kein Lernverfahren zu entwickeln, sondern lediglich ein Simulationsverfahren, während unser System die Erlernung *und* Simulation morphologischer Ableitungsregeln leisten soll.

2.1.5 JUDO

JUDO ist ein anwendungsorientiertes Softwaresystem zum computerunterstützten, linguistisch orientierten Indexing und Retrieval gegenwartsdeutscher juristischer Fachtexte.[15] Das System wird seit 1977 unter Leitung von Harald Zimmermann an den Universitäten Regensburg und Saarbrücken entwickelt.

Im folgenden werden ausschließlich diejenigen Komponenten des Systems dargestellt, die zur Simulation derivativer Wortbildungsregeln dienen, - die Erlernung derivativer Wortbildungsregeln leistet JUDO nicht. Bei der Darstellung folgen wir den Ausführungen von Dorothea Viets in dem Aufsatz

> Viets, D. (1981): Derivationsrelationen und INFO. Zwei Systemkomponenten zur Verbesserung der Retrievalergebnisse im Projekt JUDO-DS. In: Projekt JUDO-DS. Vorträge zur Beiratssitzung am 9./10./11. September 1981 in Saarbrücken. (Interne Vervielfältigung).

Die Simulation derivativer Wortbildungsregeln durch JUDO dient der Erhöhung des Recall beim Retrieval: Ziel ist es, dem Benutzer auch die Auffindung solcher relevanter Dokumente zu ermöglichen, die den von ihm mit der Suchfrage eingegebenen Deskriptor gar nicht explizit enthalten. Viets (1981: 2) erläutert dies an folgendem Beispiel: Angenommen, ein Benutzer sucht Dokumente, die sich mit der Frage

'Berechtigung zur Übermittlung von Daten'

befassen. In der Datenbank wird dann mit den Deskriptoren

'Berechtigung',
'Übermittlung' und
'Daten'

gesucht. Nun ist es wahrscheinlich, daß den Benutzer auch Dokumente interessieren, die z. B. die Wortlaute

'Zur Übermittlung von Daten berechtigen'

oder

'Berechtigung, Daten zu übermitteln'

15 Einen Überblick über JUDO gibt Zimmermann (1978: 24-31).

enthalten. Um dem Benutzer auch solche Dokumente automatisch liefern zu können, muß das System derivative Wortbildungsregeln simulieren, d. h., es muß in der Lage sein, z. B. folgende Zuordnungen zu leisten:

'berechtigen' <==> 'Berechtigung',
'übermitteln' <==> 'Übermittlung'.

Zu diesem Zweck wurden in den für das maschinelle Indexing verwendeten Thesaurus sogenannte Derivationsrelationen aufgenommen. Leider wird der Begriff 'Derivationsrelation' von Viets nicht weiter präzisiert. Wir vermuten, daß eine solche Derivationsrelation aus Pointern besteht, die Verbindungen zwischen Thesauruseinträgen herstellen, zwischen denen morphologische Ableitungsbeziehungen bestehen. Zusätzlich zu der oben bereits genannten Derivationsrelation

Verb <==> Substantiv

werden noch die 2 folgenden Typen von Derivationsrelationen eingeführt:

Substantiv <==> Adjektiv.
Beispiel: 'Nachweis' <==> 'nachweislich'.

Verb <==> Adjektiv.
Beispiel: 'nachweisen' <==> 'nachweislich'.

Beim Stand der Arbeiten im Juli 1981 verteilten sich die in den Thesaurus aufgenommenen Paare von Wörtern, zwischen denen morphologische Ableitungsbeziehungen bestehen, folgendermaßen auf diese 3 Typen von Derivationsrelationen:

Derivationsrelation	Anzahl der zu der Derivationsrelation in den Thesaurus aufgenommenen Wortpaare	
	abs.	proz.
Substantiv <==> Adjektiv	334	21,9%
Verb <==> Adjektiv	218	14,4%
Substantiv <==> Verb	966	63,8%

Abb. 2.6 (nach Viets 1981: 4)

Sucht ein Benutzer nun nach Dokumenten, die sich mit der Frage

'Berechtigung zur Übermittlung von Daten'

befassen, so wird in der Datenbank nicht nur mit den Deskriptoren

'Berechtigung',
'Übermittlung' und
'Daten'

gesucht, sondern - nachdem zu diesen Deskriptoren im Thesaurus über die Derivationsrelationen Wörter ermittelt worden sind, die zu ihnen in einer morphologischen Ableitungsbeziehung stehen - auch mit den Deskriptoren

'berechtigen' und
'übermitteln'.

Wie bereits in Abschnitt 1.1 ausgeführt, hat die Simulation derivativer Wortbildungen mit Hilfe eines Thesaurus gegenüber einem regelbasierten Verfahren, wie es in der vorliegenden Arbeit intendiert ist, den Nachteil, daß die Bildung derivativer Wortbildungen auf diejenigen Wörter beschränkt ist, die im Thesaurus durch - Derivationsrelationen repräsentierende - Pointer verknüpft sind. Dagegen leistet eine Regelmenge nicht nur die Bildung abgeleiteter Wörter zu den Erstkomponenten des Beispielkorpus, aus dem sie gewonnen wurde, sondern sie erlaubt auch die korrekte Erzeugung abgeleiteter Wörter zu weiteren, nicht im Beispielkorpus als Erstkomponenten enthaltenen Wörtern.

Der Vorteil bei der Verwendung eines Thesaurus besteht darin, daß für *alle* im Thesaurus verzeichneten Wortpaare die *korrekte* Simulation von Derivationen garantiert werden kann, während ein regelbasiertes Verfahren i. allg. einen gewissen Prozentsatz *fehlerhafter* Ableitungen, insbesondere für die Ausnahmen einer morphologischen Ableitungsbeziehung, einschließt.

2.2 Lernalgorithmen für andere sprachliche Substitutionsbeziehungen

Unsere Recherchen zum Forschungsstand haben ergeben, daß ein Algorithmus zur Erlernung morphologischer Ableitungsregeln bisher nicht existiert. Dagegen gibt es einige Lernalgorithmen für andere sprachliche Objektbereiche. Eine Auswahl dieser Verfahren soll in diesem Abschnitt vorgestellt werden. Die Auswahl besteht darin, daß wir uns speziell auf Lernalgorithmen für Substitutionsbeziehungen beschränken, da diese Verfahren dem von uns intendierten, das für eine spezielle Substitutionsbeziehung konzipiert ist, hinsichtlich der Problemstellung und teilweise auch methodisch recht ähnlich sind. Nicht in Betracht ziehen werden wir z. B. solche Verfahren, die ausgehend von einer Menge von Beispielsätzen einer Sprache, eine formale Syntax für die Erzeugung der Sätze der Sprache erlernen.[16] Der Begriff 'Substitutionsbeziehung', den wir bereits in der Einleitung von Abschnitt 2 eingeführt hatten, sei zur Verdeutlichung hier noch einmal erläutert:

> Def. *Substitutionsbeziehung*: Substitutionsbeziehungen sind alle und nur die zweistelligen Relationen, die zwischen 2 objektsprachlichen Zeichenketten bestehen gdw. sich die zweite Zeichenkette mittels einer oder mehrerer regelhafter Ersetzungen von Zeichenketten aus der ersten Zeichenkette erzeugen läßt.

Eine Substitutionsbeziehung besteht demnach beispielsweise zwischen der orthographischen Transkription deutscher Wörter einerseits und ihrer phonemischen Transkription andererseits, da phonemisch transkribierte deutsche Wörter regelgesteuert aus ihren orthographischen Entsprechungen erzeugt werden können. Eine solche regelgesteuerte Erzeugung beschreibt beispielsweise Bernd S. Müller (1981).

Bei den in diesem Abschnitt vorgestellten Projekten wurden - wie in der vorliegenden Arbeit - Algorithmen entwickelt, die zunächst in einer Lernphase aufgrund einer Analyse vorgegebener Wortpaare Regeln erlernen. In einer anschließenden Anwendungs-

16 Einige Überblicksaufsätze zu Systemen, die formale Syntaxen erlernen, sind in der Einleitung von Abschnitt 2 genannt.

phase werden dann die erlernten Regeln auf die Erstkomponenten der Paare angewendet, wobei mit den ursprünglich vorgegebenen Zweitkomponenten identische Zeichenketten erzeugt werden sollen. Die Regeln sollen allerdings mittels Generalisierung und Analogiebildung von der Lernkomponente so angelegt werden, daß sie nicht nur zu den beim Lernen vorgegebenen Erstkomponenten, sondern auch zu weiteren Erstkomponenten die korrekten Zweitkomponenten erzeugen können.

3 Arbeiten werden Gegenstand dieses Abschnittes sein: Zunächst werden wir ein von Eckhard Wolf (1977) entwickeltes Verfahren vorstellen, das u. a. Regeln zur orthographisch-phonemischen Umsetzung von Wörtern erlernt und anwendet. Der anschließend behandelte Algorithmus von Oakey/Cawthorn (1981) hat zwar dieselbe Aufgabe wie Wolfs Verfahren, er erschien uns aber dennoch für eine Darlegung unerläßlich, da er sich wesentlich anderer Methoden bedient. Das zuletzt beschriebene Verfahren mit dem Namen PELIKAN (vgl. Ring 1978) ist das einzige bekannte System, das auch morphologische Regeln erlernt. Allerdings handelt es sich dabei nicht um derivationsmorphologische, sondern um flexionsmorphologische Regeln.

2.2.1 Wolf

Eckhard Wolf hat zu Anfang der siebziger Jahre an der Universität Konstanz ein experimentelles System erarbeitet, das u. a. Regeln zur orthographisch-phonemischen Umsetzung maschinell erlernen und anwenden soll. Diese Arbeiten führte Wolf im Rahmen seiner Dissertation durch. Einen ausführlichen Bericht über seine Ergebnisse gibt er in folgendem Buch, das eine überarbeitete Fassung der Dissertation ist:

> Wolf, E. (1977): Vom Buchstaben zum Laut. Maschinelle Erzeugung und Erprobung von Umsetzautomaten am Beispiel Schriftenglisch - Phonologisches Englisch. Braunschweig.

Dieses Buch ist die Grundlage der nachstehenden Ausführungen.

Objektbereich des von Wolf entwickelten Algorithmus ist eine Menge geordneter Wortpaare, wobei die Zuordnung der Erstkomponenten dieser Paare zu den Zweitkomponenten rechtseindeutig sein

muß und wobei zwischen den 2 Wörtern jedes Paares ein natürlicher Umsetzungszusammenhang bestehen muß, der für jedes Paar innerhalb einer Menge von derselben Art ist.

> "Als natürliche Umsetzungszusammenhänge werden solche aufgefaßt, die zwischen sprachlichen Subsystemen vorgefunden werden, und zwar sowohl synchroner wie auch diachroner Art. Beispiele dafür sind: Schriftsystem - phonologisches System, phonologisches System - phonetisches System, Schriftsysteme verschiedener Sprachstufen aus dem Bereich jeweils einer Sprache." (Wolf 1977: 4f.)

Wolfs Zielsetzung ist es, eine Entdeckungsprozedur in Form eines Algorithmus zu konstruieren, die - ausgehend von einer vorgegebenen Menge kontextfreier und tentativer Anfangsregeln sowie von der Untersuchung eines Teilkorpus C_1 von Wortpaaren aus dem Objektbereich O ($C_1 \subset O$) - zunächst eine Menge von Regeln R_1 erstellt. Diese Regeln sollen dann - ausgeführt von einer anderen Komponente des Algorithmus - die Überführung der Erstkomponenten der Paare in die Zweitkomponenten leisten. Die Anfangsregeln sind mit Ausgangshypothesen über den zu beschreibenden Umsetzungszusammenhang gleichzusetzen, die im Verlauf des Lernprozesses verworfen oder verfeinert werden.

Nehmen wir beispielsweise an, das Teilkorpus C_1, auf das die Entdeckungsprozedur angewandt wird, ist eine Menge geordneter Paare, deren Erstkomponenten orthographische Transkriptionen englischer Wörter sind und deren Zweitkomponenten die entsprechenden phonemischen Transkriptionen dieser Wörter sind. Zu dieser Menge könnten u. a. die Paare ('able' <fähig>, /eibl/), ('drive' <Fahrt>, /draiv/) und ('like' <mögen>, /laik/) gehören. Die von der Entdeckungsprozedur gelieferten Regeln sollen dann, wenn sie von der Regelanwendungskomponente des Algorithmus auf die orthographischen Wörter angewandt werden, die entsprechenden phonemischen Transkriptionen liefern. Sie sollen also beispielsweise zu 'like' die phonemische Transkription /laik/ liefern.

Die Regeln sollen allerdings nicht nur die richtige Umsetzung der Erstkomponenten des Korpus C_1 leisten, aus dem sie gewonnen wurden, sondern sie sollen auch für die Mehrzahl der Erstkomponenten des gesamten Objektbereiches O fehlerfreie Umsetzungen

liefern. Im Beispiel sollen sie also nicht nur zu den orthographischen Transkriptionen des Teilkorpus, sondern auch zur Mehrzahl der nicht im Teilkorpus enthaltenen orthographischen Transkriptionen englischer Wörter die korrekten phonemischen Transkriptionen liefern. Eine Betonung liegt hierbei auf 'Mehrzahl', denn es kann nicht angestrebt werden, eine fehlerfreie Umsetzung für *alle* Erstkomponenten des Objektbereiches zu liefern. Da es bei der Beschreibung von Umsetzungszusammenhängen im Bereich natürlicher Sprachen, wie es hier der Fall ist, wegen der Unendlichkeit des Objektbereiches

> "prinzipiell nicht möglich scheint, angeben zu können, ob und wann der Gegenstand unter bestimmten Zielsetzungen hinreichend von einer Beschreibung erfaßt ist, kann eine Beschreibung im allgemeinen nur als approximativ angesehen werden." (Wolf 1977: 9)

Ist nun die aus C_1 gewonnene Regelmenge R_1 bei der Anwendung auf O nicht ausreichend adäquat[17], so extrahiert Wolfs Verfahren aus O ein zu C_1 disjunktes Teilkorpus C_2 und ermittelt aus C_2 eine weitere Regelmenge R_2', die es mit der aus C_1 gewonnenen Regelmenge R_1 zu einer neuen Regelmenge R_2 verbindet. R_2 wird daraufhin auf O angewendet und hinsichtlich ihrer Adäquatheit bewertet. Ist R_2 ausreichend adäquat, so wird das Verfahren beendet. Ist R_2 nicht ausreichend adäquat, so wird der beschriebene Regelerweiterungsprozess so lange fortgesetzt, bis schließlich eine auf der Untersuchung von disjunkten Teilkorpora C_1, ..., C_n beruhende Regelmenge R_n gefunden ist, die den im Objektbereich geltenden Umsetzungszusammenhang ausreichend adäquat beschreibt. Da der Algorithmus selbst entscheidet, ob eine Regelmenge eine adäquate Beschreibung leistet, befähigt diese Arbeitsweise den Algorithmus, selbst ein für den gesamten Objektbereich repräsentatives Teilkorpus

$$T = \bigcup_{i=1}^{n} C_i$$

zu finden: Ein Teilkorpus T ist nämlich genau dann für den

17 Wolfs Adäquatheitsbegriff wird im folgenden noch näher erläutert. Er unterscheidet sich von dem in der vorliegenden Arbeit (in Abschnitt 3.2) gewählten Adäquatheitbegriff.

Objektbereich repräsentativ, wenn die aus 7 gewonnene Regelmenge den im Objektbereich gegebenen Umsetzungszusammenhang adäquat beschreibt.

Wolf beansprucht nicht, daß sein Algorithmus Strukturmodell für die menschliche Fähigkeit ist, Umsetzungszusammenhänge zu beschreiben, d. h., daß er kognitive Prozesse zur Beschreibung oder Erlernung von Umsetzungszusammenhängen modelliert. Sein Algorithmus soll vielmehr Leistungsmodell sein, die Operationen des Algorithmus sollen den jeweiligen Operationen des Menschen "nur insoweit entsprechen, als Ausgangs- und Zielobjekt identisch sind." (Wolf 1977: 9)

Für die Beschreibung von Umsetzungszusammenhängen verwendet Wolf den Regeltyp

A → B/<C,D>.

Dieser Regeltyp ist folgendermaßen zu lesen: Wenn in der Erstkomponente das *Einzel*zeichen A zwischen den - möglicherweise auch leeren - Zeichenketten C und D auftritt, so ist in der Zweitkomponente anstelle von A die - möglicherweise leere - Zeichen*kette* B zu notieren. Eine Regel zur Umsetzung von 'i' in /ai/, wie sie bei der Phonemisierung von englisch 'like' erforderlich wäre, könnte dann beispielsweise lauten:

'i' → 'ai'/<'','k'>.

Mehrere Regeln dieses Typs werden so zu *geordneten* Regelmengen zusammengefaßt, daß die Regeln mit den spezielleren Kontextbedingungen (Kontextbedingungen sind die Zeichenketten C und D) stets *vor* solchen mit den allgemeineren Kontextbedingungen stehen. Bei der Anwendung der Regeln auf ein Wort werden dann grundsätzlich die weiter vorne stehenden, also die spezielleren Regeln *vor* den weiter hinten stehenden, also den allgemeineren Regeln anzuwenden versucht. Hierdurch wird garantiert, daß jede Regel potentiell zur Anwendung kommen kann. (Würde eine allgemeinere vor einer spezielleren Regel anzuwenden versucht, so käme die speziellere Regel niemals zur Anwendung!)

Wolfs Algorithmus besitzt - sehr vereinfacht dargestellt -

folgenden Aufbau:

- *Eingabe* ist eine Menge von Wortpaaren, genannt Objektbereich O und eine Menge kontextfreier Anfangsregeln.
- Ein *Materialaufbereitungsverfahren* extrahiert jedesmal, wenn es aufgerufen wird, eine Teilmenge C_i aus O und übergibt diese Teilmenge
  - entweder als Grundlage für die Erlernung des Umsetzungszusammenhanges an das Analyseverfahren (s. u.)
  - oder als Grundlage für die Überprüfung verschiedener Adäquatheitsstufen einer zuvor gewonnenen Regelmenge an das Prüfverfahren (s. u.).

  Wird das Materialaufbereitungsverfahren mehrmals bei einem Erlernungsprozess zur Extraktion einer Teilmenge aufgerufen, so muß gelten, daß jede neu extrahierte Teilmenge C_i disjunkt ist von der Vereinigungsmenge der zuvor extrahierten Teilmengen $C_1, \ldots, C_{i-1}$.
- Das *Analyseverfahren* wird im Anschluß an das Materialaufbereitungsverfahren durchlaufen. Dieses Verfahren konstruiert u. a. zu einer vom Materialaufbereitungsverfahren übergebenen Menge C eine geordnete Menge R_A von Regeln. Diese Regelmenge ist ziemlich redundant, d. h., es lassen sich Regeln aus dieser Menge zu einer Regel zusammenfassen und es lassen sich Kontextbedingungen in den Regeln kürzen, ohne daß der durch R_A beschriebene Umsetzungszusammenhang verändert wird.
- Aufgabe des sich anschließenden *Syntheseverfahrens* ist es daher, zu R_A ein redundanzärmeres Regelsystem zu erzeugen, das denselben Umsetzungszusammenhang wie R_A beschreibt. Eine weitere Aufgabe des Syntheseverfahrens, die eng mit der Redundanzenbeseitigung in R_A verknüpft ist, besteht in der Vereinigung des aus R_A entstehenden redundanzärmeren Regelsystems mit der vorgegebenen Menge kontextfreier Anfangsregeln oder mit einem in vorangehenden Durchläufen entstandenen, nicht adäquaten Regelsystem R_N zu einem neuen Regelsystem R_S.
- Das *Prüfverfahren* hat zusammen mit dem sich anschließenden Bewertungsverfahren (s. u.) u. a. die Aufgabe, den Grad der C-Adäquatheit und der schwachen O-Adäquatheit einer vom Syn-

theseverfahren übergebenen Regelmenge zu überprüfen. Eine aus einem Korpus C_i gewonnene Regelmenge R nennt Wolf

- *C-adäquat* gdw. R nicht mehr als X% der in den Paaren von C_i gegebenen Erstkomponenten falsch umsetzt. X ist dabei eine vom Benutzer in Abhängigkeit vom Erkenntnisinteresse vorzugebende Toleranzgrenze.
- *schwach O-adäquat* gdw. R nicht mehr als X% der in den Paaren von C_{i+1} gegebenen Erstkomponenten falsch umsetzt.

Neben der *C*-Adäquatheit und der schwachen *O*-Adäquatheit zieht Wolf noch eine sogenannte *starke O-Adäquatheit* in Betracht, auf die hier allerdings nicht näher eingegangen werden soll. Der Grad der starken *O*-Adäquatheit ergibt sich aus dem Umsetzungsverhalten der Regelmenge R bei ihrer Anwendung auf die Erstkomponenten der Paare in den Korpora C_{i+2}, ..., C_{i+n}, wobei n eine durch den Benutzer in Abhängigkeit vom Erkenntnisinteresse vorzugebende Größe ist.

Der Beitrag des Prüfverfahrens zur Überprüfung der verschiedenen Adäquatheitsstufen besteht darin, daß es die vom Syntheseverfahren gelieferten Regeln auf die Erstkomponenten der Paare des Korpus C_i oder des Korpus C_{i+1} anwendet, die Ergebnisse der Anwendung mit den in den Korpora gegebenen Zweitkomponenten vergleicht und den prozentualen Anteil fehlerhafter Umsetzungen an der Gesamtzahl der Umsetzungen berechnet.

Ist die Umsetzung falsch, so wird ein Tripel, bestehend aus der betreffenden Erstkomponente (= Erstkomponente des Tripels), der vorgegebenen Zweitkomponente (= Zweitkomponente des Tripels) und der falschen Umsetzung (= Drittkomponente des Tripels) gebildet und in eine Menge 7 solcher Tripel eingeordnet. Diese Menge wird später u. U. von einem Korrekturverfahren weiterverarbeitet.

- An das Prüfverfahren schließt sich das *Bewertungsverfahren* an: Das Bewertungsverfahren stellt u. a. den Grad der *C*-Adäquatheit und der schwachen *O*-Adäquatheit einer vom Syntheseverfahren gelieferten Regelmenge fest, indem es den vom Prüfverfahren gelieferten Wert für den prozentualen Anteil fehlerhafter Umsetzungen mit der vom Benutzer vorgegebenen Toleranz-

grenze[18] vergleicht. Der weitere Arbeitsverlauf des Algorithmus hängt vom Bewertungsergebnis ab.

Im einzelnen verläuft das Bewertungsverfahren folgendermaßen:

- Zunächst wird die vorliegende Regelmenge auf *C*-Adäquatheit überprüft.

  Ist sie nicht *C*-adäquat, so wird das im folgenden noch zu erläuternde Modifikationsverfahren aufgerufen, das aus den vom Prüfverfahren gelieferten fehlerhaften Tripeln neue, korrigierte Eingabe*paare* für das Analyseverfahren erzeugt.

  Ist die Regelmenge dagegen *C*-adäquat, so wird das Materialaufbereitungsverfahren aufgerufen, das das Korpus C_{i+1} aus *O* extrahiert und zur Überprüfung der schwachen *O*-Adäquatheit an das Prüfverfahren übergibt.

- Ist die Regelmenge nicht schwach *O*-adäquat, so wird das Korpus C_{i+1} an das Modifikations- und anschließend an das Analyseverfahren übergeben und aufgrund der Analyse von C_{i+1} und der bereits bestehenden Regelmenge eine neue Regelmenge konstruiert, die dann zunächst wieder auf *C*-Adäquatheit getestet wird, wobei *C* dann gleich dem früheren C_{i+1} ist.

  Ist die Regelmenge dagegen schwach *O*-adäquat, so schließt sich die Überprüfung der starken *O*-Adäquatheit an, die hier jedoch nicht dargestellt werden soll.

- Wird eine Regelmenge durch das Bewertungsverfahren als nicht ausreichend *C*-adäquat oder als nicht ausreichend schwach *O*-adäquat bewertet, so wird das *Modifikationsverfahren* aufgerufen. Das Modifikationsverfahren schafft die Voraussetzung für die Konstruktion einer verbesserten Regelmenge, indem es ausgehend von der vom Prüfverfahren gelieferten Menge *T*[19] eine Menge geordneter Paare liefert, deren Erstkomponenten identisch sind mit den Erstkomponenten der Tripel aus *T* und deren Zweitkomponenten aufgrund eines Vergleichs der fehlerhaften Drittkomponenten der Tripel in *T* mit den Zweitkomponen-

18 Vgl. hierzu die obige Erläuterung des Prüfverfahrens.
19 Vgl. dazu die obige Beschreibung des Prüfverfahrens.

ten der Tripel in 7 gebildet werden. Diese neuen Zweitkomponenten unterscheiden sich von den Zweitkomponenten in 7 lediglich durch eine andere Segmentierung und eine eventuelle Einfügung von leeren Ketten (= Nullstrings). Die vom Modifikationsverfahren gelieferten Paare werden an das Analyseverfahren weitergegeben, das auf ihrer Basis neue Regeln liefert, die vom Syntheseverfahren mit der bereits bestehenden nicht-adäquaten Regelmenge zu einer neuen Regelmenge integriert werden.

Wolf hat sein Verfahren exemplarisch angewendet zur Entdeckung von Umsetzungszusammenhängen, die zwischen den orthographisch transkribierten englischen Wörtern und den entsprechenden phonemisch transkribierten Wörtern bestehen. Bei dieser Anwendung gab er 26 kontextfreie Anfangsregeln vor, die tentative Phonemisierungen für die 26 Buchstaben des lateinischen Alphabets festlegen. Das zugrunde gelegte Gesamtkorpus wurde nach Häufigkeitskriterien zusammengestellt und besteht aus 2.800 verschiedenen Paaren, deren Erstkomponente jeweils ein orthographisch transkribiertes und deren Zweitkomponente das entsprechende phonemisch transkribierte englische Wort ist. Dieses Gesamtkorpus wurde schrittweise bearbeitet, indem zunächst 30 Teilkorpora zu je 40 Paaren durch das Materialaufbereitungsverfahren ausgefiltert wurden und auf ihrer Basis jeweils sukzessive eine Regelmenge konstruiert wurde. Diese 30 Teilkorpora enthalten zusammen 1.200 Wortpaare. Die restlichen 1.600 Wortpaare wurden anschließend in 16 Korpora zu je 100 Paaren verarbeitet. Von den aus diesen 46 Korpora resultierenden 46 aufeinander aufbauenden Regelmengen waren zwar alle *C*-adäquat, jedoch war keine schwach *O*-adäquat. die Toleranzgrenze bei der Bewertung der Adäquatheit betrug 10%, d. h., um die jeweilige Adäquatheitsstufe zu erreichen, mußten mindestens 90% der orthographischen Wörter korrekt phonemisiert werden. Wolfs Anwendungsergebnisse zeigen, daß der Grad der schwachen *O*-Adäquatheit der Regelmengen im Durchschnitt zunimmt: Eine durch die Verarbeitung von 1.000 Eingabepaaren aus einer zuvor vorhandenen Regelmenge entstandene Regelmenge setzte *im Durchschnitt* 10% mehr Erstkomponenten in die richtigen Zeichenketten um als die ursprüngliche Regelmenge.

Im folgenden sei eine Auswahl von Wolfs Ergebnissen für den Grad der schwachen *O*-Adäquatheit wiedergegeben:[20]

Nummer des Korpus, aus dem die Regelmenge gewonnen wurde	Prozentualer Anteil richtig umgesetzter Erstkomponenten an der Gesamtzahl umgesetzter Erstkomponenten bei Anwendung der Regelmenge auf das Folgekorpus
5	7,5%
10	20,0%
15	22,5%
20	20,0%
25	20,0%
30	9,0%
35	16,0%
40	28,0%
45	26,0%

Abb. 2.7 (nach Wolf 1977: 157)

Diese Ergebnisse sind insgesamt nicht sehr zufriedenstellend. Zumindest stellen sie die praktische Anwendbarkeit des Algorithmus in Frage. So scheint der Lernalgorithmus etwa für die maschinelle Erzeugung eines Phonemisierungsalgorithmus, der orthographische Texte für die Steuerung eines Sprachsynthesegerätes aufbereitet, nicht geeignet zu sein.

Abschließend seien noch 2 Merkmale von Wolfs Verfahren besonders hervorgehoben, die uns generell für Lernalgorithmen zu Substitutionsbeziehungen besonders erstrebenswert erscheinen:

- Der Algorithmus ist *sprachunabhängig*, d. h., er erlernt Umsetzungszusammenhänge auf Wortebene in *beliebigen* Sprachen (vgl. Wolf 1977: 12f.).
- Der Regeltyp zur Beschreibung und Simulation von Umsetzungszusammenhängen ist so gewählt, daß eine von der Entdeckungsprozedur zu einem Teilkorpus erzeugte Regelmenge nicht nur den in diesem *Teilkorpus* vorliegenden Umsetzungszusammenhang adäquat simulieren kann, vielmehr kann sie potentiell auch den im *gesamten Objektbereich* vorliegenden Umsetzungszusammen-

20 Die Verbesserung des Grades der schwachen *O*-Adäquatheit um *durchschnittlich* 10% bei der Verarbeitung von 1.000 Eingabepaaren läßt sich in dieser *Auswahl* von Ergebnissen noch nicht erkennen.

hang adäquat simulieren.

Ein Regeltyp etwa der Form A → B, wobei A jeweils Erstkomponente und B Zweitkomponente eines zum Lernen vorgegebenen Wortpaares ist und der A durch B ersetzt, besäße diese Eigenschaft nicht. Eine Menge von Regeln dieses Typs würde lediglich die Aufsuche in einem Lexikon simulieren, dessen Lemmata jeweils die Erstkomponenten und dessen Informationsteile jeweils die Zweitkomponenten der zum Lernen vorgegebenen Wortpaare sind. Die Regelmenge könnte also zu keinen anderen als den zum Lernen vorgegebenen Erstkomponenten die entsprechenden Zweitkomponenten erzeugen.

Nach der Lektüre der obigen Darstellung von Wolfs Verfahren vermutet man vielleicht, daß sein Verfahren nicht nur für die Erlernung orthographisch-phonemischer Umsetzungsregeln geeignet ist, sondern auch zur Erlernung und Simulation morphologischer Ableitungsregeln, und somit die Konstruktion des von uns intendierten Algorithmus überflüssig ist. Dies ist jedoch nicht der Fall. Wolfs Algorithmus hätte bei seiner Anwendung zur Erlernung und Simulation morphologischer Ableitungsregeln mindestens folgende Mängel:

- Die konstruierten Regelmengen wären *redundant*, da sie *jedes* Einzelzeichen einer vorgegebenen Wortform bei der Bildung einer entsprechenden abgeleiteten Wortform substituieren. Bei der Simulation morphologischer Ableitungen muß in den meisten Fällen nur ein Teil der Zeichen jeder Wortform substituiert werden.
- Weiterhin würde Wolfs Regeltyp morphologische Ableitungsregeln *nicht anschaulich* beschreiben, da jede Regel dieses Typs stets nur die Ersetzung von *Einzel*zeichen der vorgegebenen Wortform durch Zeichen*ketten* der zu erzeugenden Wortform darstellen kann. Für eine anschauliche Beschreibung sind jedoch solche Regeln wünschenswert, die auch die Ersetzung von Zeichen*ketten* der vorgegebenen Wortform durch Zeichen*ketten* der zu erzeugenden Wortform darstellen können.
- Schließlich existiert kein Mechanismus zur Verhinderung morphologischer Ableitungen. Wie wir in Abschnitt 3.1 noch sehen werden, ist ein solcher Mechanismus bei der Simulation morphologischer Ableitungsregeln jedoch erforderlich.

2.2.2 Oakey/Cawthorn

Das in diesem Abschnitt beschriebene System wurde von S. Oakey und R. C. Cawthorn zu Beginn der achtziger Jahre an der Teesside Polytechnic/Middlesbrough (England) entwickelt. Über das System wird in folgendem Aufsatz berichtet, den wir unseren Ausführungen zugrunde legen:

> Oakey, S./Cawthorn, R. C. (1981): Inductive learning of pronunciation rules by hypothesis testing and correction. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence. August 1981. Bd. 1. S. 109-114.

Oakey und Cawthorn haben dasselbe Ziel wie Wolf (vgl. Abschnitt 2.2.1): Gegenstand ihres Projektes ist die Entwicklung eines Algorithmus, der aus vorgegebenen Beispielwortpaaren Regeln erlernt. Ein Wortpaar besteht dabei aus einer orthographischen Transkription und einer phonemischen Transkription desselben Wortes. Die Regeln sollen beschreiben, wie die phonemische Transkription aus der orthographischen Transkription erzeugt wird. Sie sollen möglichst allgemein formuliert sein, so daß sie bei ihrer späteren maschinellen Anwendung auch zu weiteren, nicht zum Lernen vorgegebenen orthographischen Wörtern die entsprechenden phonemischen Wörter erzeugen können.

Die zu erlernenden Regeln sind vom Lernalgorithmus in der allgemeinen Form

(Graph(kette) (linker Kontext) (rechter Kontext) (Phonem(kette)))

darzustellen. Diese Regelform ist zu lesen: Tritt die Graphkette in einem vorgegebenen orthographischen Wort unmittelbar rechts vom linken und unmittelbar links vom rechten Kontext auf, so ist sie durch die Phonemkette zu ersetzen.

Der Regeltyp ist nicht etwa - wie es zunächst vielleicht scheinen mag - lediglich eine andere Notation des von Wolf verwendeten Regeltyps. Abgesehen davon, daß der Regeltyp auch die Ersetzung von Graph*ketten* und nicht nur von *einzelnen* Graphen erlaubt, lassen Oakey und Cawthorn i. U. zu Wolf in den Kontextbedingungen nicht nur Graph(-kett-)en zu, sondern auch Bezeichnungen für *Klassen* von Graph(-kett-)en, wie etwa CONS1 für alle Graphketten der Länge 1, die aus 'konsonantischen' Graphen beste-

hen. Wie bei Wolf sind die Regelmengen so geordnet, daß speziellere Regeln stets vor allgemeineren Regeln anzuwenden versucht werden.

Die Lernstrategie des Verfahrens von Oakey und Cawthorn unterscheidet sich erheblich von der in Wolfs System:

Zu Beginn eines Lernprozesses werden dem Lernalgorithmus 26 kontextfreie Anfangsregeln vorgegeben, die für jeden der 26 Buchstaben des lateinischen Alphabets angeben, durch welches Phonemzeichen er ersetzt werden soll. Diese Regeln sind mit Ausgangshypothesen über den zu beschreibenden Umsetzungszusammenhang gleichzusetzen, die im Verlauf des Lernprozesses verworfen oder verfeinert werden. Im allgemeinen wird es sicher notwendig sein, die 26 Anfangsregeln zu verändern, da in den meisten - wenn nicht in allen - Sprachen mit Alphabetschrift die Buchstaben je nach Kontext *verschiedene* Lautwerte repräsentieren. Um die Anfangsregeln schrittweise zu verbessern, arbeitet das Verfahren im wesentlichen mit folgenden Teiloperationen:

- *Regelerzeugung*: Die vorliegende Regelmenge wird auf ein orthographisches Wort angewendet und dabei wird ein phonemisches Wort erzeugt (Vor der Anwendung auf das erste orthographische Wort handelt es sich bei der Regelmenge um die 26 Anfangsregeln, später um daraus hervorgegangene verbesserte Regelmengen). Das erzeugte Wort wird dann mit dem entsprechenden vorgegebenen (korrekten) phonemischen Wort verglichen. Bei Identität werden die Regeln nicht verändert, und es wird das nächste graphische Wort verarbeitet. Sind das erzeugte und das vorgegebene phonemische Wort dagegen nicht identisch, so bedeutet dies, daß die vorliegende Regelmenge nicht korrekt ist. In diesem Fall werden zunächst sehr spezielle Regeln konstruiert. Diese sogenannten *Basisregeln* lassen sich nur auf das falsch umgesetzte orthographische Wort anwenden, da als linker und rechter Kontext jeweils die Zeichenketten zwischen der umzusetzenden Graphkette und den Wortgrenzen genommen werden. Die Basisregeln beschreiben die korrekte Umsetzung der zunächst falsch umgesetzten Graphen.
- *Generalisierung*: Jedesmal wenn eine Basisregel erzeugt wurde, wird diese mit den bereits vorliegenden Regeln verglichen.

Finden sich dabei Regeln, die *dieselbe* Ersetzung unter ähnlichen oder identischen Kontextbedingungen beschreiben, so werden diese Regeln mit der neuen Basisregel zu einer *allgemeineren generalisierten Regel* synthetisiert. Die Generalisierung besteht dabei in einer Verallgemeinerung der beiden Kontextbedingungen. So werden beispielsweise die beiden Basisregeln

(CK (D U) () (K))
(CK (L U) () (K)),

die aus den Wortpaaren

('duck' <Ente>, /dʌk/)
('luck' <Glück>, /lʌk/)

gewonnen wurden, mittels Generalisierung zu der allgemeineren Regel

(CK (CONS1 U) () (K))

zusammengefaßt. Der Generalisierungsprozess verläuft rekursiv, d. h., mehrere, bereits generalisierte Regeln können wiederum zu einer (weiter) generalisierten Regel zusammengefaßt werden usw. Um später etwaige Übergeneralisierungen rückgängig machen zu können, werden jeder generalisierten Regel die Regeln, aus denen sie hervorgegangen ist, als Zusatzinformationen beigefügt und in der bereits vorliegenden Regelmenge gelöscht, bzw. - im Falle einer gerade neu gebildeten Basisregel - gar nicht erst in die Regelmenge aufgenommen.

- *Beseitigung von Regelkollisionen*: Stellt sich bei einer neu erzeugten Regel heraus, daß in der Regelmenge bereits eine Regel existiert, die *dieselbe* Graphkette unter *gleichen* Kontextbedingungen durch eine *andere* Phonemkette ersetzt, so heißt dies, daß sich diese 2 Regeln widersprechen. In einem solchen Fall wird diejenige der beiden Regeln in die Regelmenge aufgenommen, die für die meisten der vorgegebenen orthographischen Wörter gilt.
- *Einsortierung neuer Regeln in die Regelmenge*: Neu erzeugte generalisierte Regeln oder neu erzeugte nicht generalisierbare Basisregeln werden so in die Menge bereits vorliegender Regeln eingeordnet, daß bei der späteren Regelanwendung die-

jenigen Regeln mit den spezielleren Kontextbedingungen stets vor denen mit den allgemeineren Kontextbedingungen anzuwenden versucht werden.

- *Regelstabilisierung*: Unter dieser irreführenden Bezeichnung verbirgt sich die Beseitigung von Übergeneralisierungen: Nachdem mit den oben genannten Teilschritten eine Menge von Beispielwortpaaren abgearbeitet und dabei eine Regelmenge erzeugt worden ist, wird in einer abschließenden Kontrolle diese Regelmenge auf die graphischen Wörter aller Beispielwortpaare angewendet, und es werden die dabei erzeugten phonemischen Wörter mit den entsprechenden vorgegebenen phonemischen Wörtern verglichen. Aufgrund dabei ggf. auftretender Nichtübereinstimmungen lassen sich unter den generalisierten Regeln Übergeneralisierungen ausfindig machen. Diese übergeneralisierten Regeln werden unter Rückgriff auf die spezielleren Regeln, aus denen sie synthetisiert wurden und die ihnen noch als Zusatzinformationen beigegeben sind, in speziellere Regeln aufgespalten.

Oakey und Cawthorn haben ihr Verfahren lediglich unter dem Aspekt der C-Adäquatheit (im Sinne Wolfs) der erzeugten Regelmengen evaluiert. Die Umsetzungsergebnisse waren dabei im Mittel gleich gut wie die Umsetzungsergebnisse mit einer Regelmenge, die von einer anderen Forschergruppe intellektuell erstellt wurde. Zur Evaluierung wurden 3 Korpora herangezogen:

- die 477 häufigsten Wörter aus dem Bereich englischer Jugendliteratur,
- alle mit dem Buchstaben 'A' beginnenden Einträge aus einem englischen Aussprachewörterbuch,
- alle mit dem Buchstaben 'B' beginnenden Einträge des Aussprachewörterbuches.

Den Wörtern jedes dieser 3 Korpora wurden die phonemischen Transkriptionen intellektuell beigegeben. Sodann wurde für jedes der 3 Korpora ein Lernlauf durchgeführt. Aus den 3 dabei erzeugten Regelmengen wurden alle Basisregeln, d. h. alle nicht generalisierten Regeln, entfernt. Jede dieser reduzierten Mengen wurde sodann auf die orthographischen Wörter des Korpus, aus dem sie gewonnen wurde, angewandt und es wurde ermittelt, wieviele der

erzeugten phonemisierten Wörter und der darin enthaltenen Phoneme *richtig* erzeugt wurden, d. h. mit den vorgegebenen phonemischen Wörtern und darin enthaltenen Phonemen übereinstimmen. Dieselbe Untersuchung wurde mit der intellektuell erstellten Regelmenge durchgeführt. Dabei kam folgendes Ergebnis zustande:

Korpus	Gesamtzahl	Prozentualer Anteil korrekter Umsetzungen bei *intellektuell* erstellter Regelmenge	Prozentualer Anteil korrekter Umsetzungen bei *automatisch* erstellter Regelmenge
477 häufigste Wörter			
Phoneme	1.745	91%	84%
Wörter	477	84%	59%
Wörterbucheinträge mit dem Anfangsbuchstaben 'A'			
Phoneme	6.378	64%	74%
Wörter	1.015	16%	21%
Wörterbucheinträge mit dem Anfangsbuchstaben 'B'			
Phoneme	4.630	79%	85%
Wörter	866	47%	43%

Abb. 2.8 (nach Oakey/Cawthorn 1981: 113)

Diese Statistik läßt zunächst vielleicht vermuten, daß der Algorithmus von Oakey/Cawthorn hinsichtlich der *C*-Adäquatheit schlechter abschneidet als der von Wolf. Bei der Evaluierung von Wolfs Algorithmus waren alle generierten Regelmengen *C*-adäquat, d. h., sie erzeugten zu mindestens 90% der orthographischen Wörter des zum Lernen verwendeten Korpus die korrekten phonemischen Wörter. Berücksichtigt man, daß Oakey/Cawthorn i. U. zu Wolf vor der Evaluierung die Basisregeln aus der Regelmenge entfernt haben und daß bei Belassen dieser Regeln in der Regelmenge "the statistics would always show 100% correctness of pronunciation" (Oakey/Cawthorn 1981: 113), so schneidet das Verfahren von Oakey und Cawthorn hinsichtlich der *C*-Adäquatheit gleich gut ab wie das von

Wolf.

Evaluierungsergebnisse zur schwachen und zur starken *O*-Adäquatheit (im Sinne von Wolf) legen Oakey und Cawthorn bedauerlicherweise nicht vor. Gerade diese Evaluierungsergebnisse wären für die Beurteilung der praktischen Verwendbarkeit ihres Systems von besonderem Interesse.

Der inhaltliche Vergleich der intellektuell und der automatisch erzeugten Regeln ergab: "A number of rules produced are exactly the same as some in the manually produced set, while others show considerable similarity." (Oakey/Cawthorn 1981: 114)

Ansonsten weist der beschriebene Algorithmus ähnliche positive Eigenschaften auf wie der von Wolf:

- Er ist sprachunabhängig.
- Der zur Beschreibung von Umsetzungszusammenhängen gewählte Regeltyp erlaubt es, den zu beschreibenden Umsetzungszusammenhang nicht nur für die Paare des zum Lernen vorgegebenen Teilkorpus korrekt zu beschreiben, sondern auch für weitere Paare der Grundgesamtheit, aus der das Teilkorpus stammt.
- Der Regeltyp erlaubt (i. U. zu dem von Wolf) Klassenbildungen innerhalb von Kontextbedingungen. Dadurch werden die erzeugten Regelapparate für den Benutzer leichter überschaubar.

Eine etwaige Verwendung des dargestellten Algorithmus zur Erlernung und Simulation morphologischer Ableitungsregeln kommt ebenfalls aus ähnlichen Gründen nicht in Betracht wie Wolfs Verfahren:

- Der Algorithmus würde Regelmengen konstruieren, die *jedes* Zeichen einer vorgegebenen Wortform ersetzen, um eine entsprechende abgeleitete Wortform zu erzeugen. Bei der Simulation morphologischer Ableitungen muß in der Regel *nur ein Teil* der Zeichen ersetzt werden.
- Oakey/Cawthorns Verfahren weist keine Komponente zur Verhinderung morphologischer Ableitungen auf.

2.2.3 PELIKAN

PELIKAN (= *P*rogrammsystem zum *E*rlernen *l*inguistischer *K*lassifikationsalgorithmen durch *A*nalogieschlüsse) ist ein Lernalgorithmus, der bei der Fa. Siemens/München als Teilkomponente des Informationssystems CONDOR (= *C*ommunication in *n*atürlicher Sprache mit *d*ialogorientiertem *R*etrievalsystem) in der zweiten Hälfte der siebziger Jahre entwickelt wurde.[21] Über den Algorithmus wird in dem Aufsatz

> Ring, H. (1978): PELIKAN - ein Lernsystem für linguistische Klassifikationsalgorithmen. In: Nachrichten für Dokumentation 6. S. 224-226.

berichtet.

> "PELIKAN findet auf Grund von eingegebenen Musterbeispielen allgemeine Strukturregeln und ermöglicht es so, linguistische Analysealgorithmen automatisch zu erstellen und zu warten." (Ring 1978: 225)

Die Musterbeispiele sind geordnete Paare. Die Komponenten dieser Paare können Vertreter beliebiger Typen linguistischer Einheiten sein. Jedoch müssen *innerhalb einer* Menge geordneter Paare, die dem Programm zur Bearbeitung übergeben wird, *alle* Erstkomponenten Vertreter *desselben* Typs sein und ebenso *alle* Zweitkomponenten. Weiterhin muß es möglich sein, die Zweitkomponente jedes Paares als Funktion eines linken Teilstrings der Erstkomponente oder der invertierten Erstkomponente zu beschreiben.

Ring führt als Beispiele folgende mögliche Typen linguistischer Einheiten an, deren Vertreter als Komponenten dieser Paare eingesetzt werden können:

Erstkomponente	Zweitkomponente
Vorname	Sexus
Vollform	Lemma
Vollform	Wortart

Abb. 2.9

PELIKAN erlernt nun aus diesen Paaren Regeln, die in ihrer

21 Einen Überblick über CONDOR geben Wieland/Haller (1978).

Gesamtheit als Analysealgorithmus bezeichnet werden und die zur Speicherplatzersparnis in Form eines sogenannten Analysebaumes gespeichert werden. Diese Regeln liefern bei späterer Anwendung auf die Erstkomponenten der Paare die entsprechenden Zweitkomponenten. Dabei prüft jede Regel stets die Erstkomponente, bzw. in einigen Fällen die invertierte Erstkomponente, auf das Vorliegen eines bestimmten linken Teilstrings. In Abhängigkeit vom Vorliegen dieses Teilstrings wird die entsprechende Zweitkomponente gebildet.

Ring verdeutlicht die Arbeitsweise von PELIKAN an einem Beispiel:

> "Es soll ein Programm erzeugt werden, das für beliebige Vornamen entscheidet, ob es sich um männliche (M) oder weibliche (W) Namen handelt.
>
> Hierzu geben wir dem System PELIKAN die Anfangshypothese 'M' ein, d. h., es wird nun jeder Name als männlich eingestuft. Geben wir jetzt dem System die Information ('Eva', W), so stellt es fest, daß auf Grund seiner Anfangshypothese dem Namen 'Eva' die Klasse M zuzuordnen wäre. Die neue Information zeigt aber, daß dies falsch ist. Daher ändert das System seine Hypothese. Es speichert aber nicht etwa den vollständigen Namen 'Eva' ab, sondern nur die Hypothese, daß Wörter, die auf 'a' enden, der Klasse W angehören, alle übrigen aber der Klasse M." (Ring 1978: 225)

Dabei entsteht der obere Analysebaum in Abb. 2.10.

> "Geben wir nun beispielsweise die Information ('Maria', W) ein, so läßt das System den Analysebaum unverändert, da dieses Ergebnis auch durch die augenblickliche Hypothese zugeordnet würde. Dagegen ergibt die Information ('Elke', W) eine weitere Modifikation des Analysebaumes, wie sie im mittleren Teil von Abb. 2 (gemeint ist hier Abb. 2.10; Ergänzung des Verf.) dargestellt ist.
>
> Wenn wir nun das System nach dem Geschlecht des Namens 'Uwe' fragen, so wird es mit W antworten, da 'Uwe' ebenso wie 'Elke' auf 'e' endet. Die Eingabe ('Uwe', M) veranlaßt jedoch eine weitere Verfeinerung des Analysebaums (Abb. 2, unterer Teil (gemeint ist der untere Teil von Abb. 2.10; Ergänzung des Verf.)), so daß alle dem System bisher eingegebenen Namen ('Eva', 'Maria', 'Elke', 'Uwe') richtig zugeordnet werden können. Darüber hinaus kann das System jetzt aber auch noch einen Großteil der übrigen Vornamen richtig klassifizieren (z. B. 'Ute', 'Otto', 'Manfred', 'Erika')." (Ring 1978: 225)

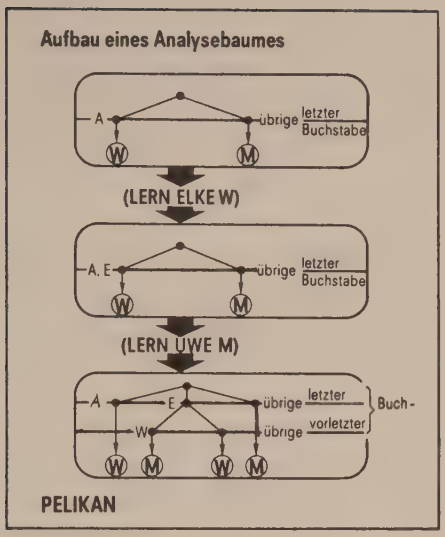


Abb. 2.10 (aus Ring 1978: 225)

Innerhalb der linguistischen Analyse des Systems CONDOR wurde PELIKAN eingesetzt zur Erlernung eines Algorithmus, der zu Vollformen der deutschen Gegenwartssprache die Wortart bestimmt.

Die Vorteile, die Ring in einem lernenden System, wie PELIKAN, gegenüber einem nicht-lernenden System sieht, sind etwa dieselben, die wir bereits in Abschnitt 1.1 genannt hatten:

- Der Arbeitsaufwand des Benutzers oder Programmierers bei der Erstellung von linguistischen Analysealgorithmen reduziert sich auf die Eingabe von Beispielen, aus denen gelernt werden soll.
- Das lernende Programm kann Konsistenzprüfungen bei der Erstellung des Analysealgorithmus durchführen.

Insgesamt ist Rings Darstellung zu kurz, um hier einen Vergleich von PELIKAN mit dem in der vorliegenden Arbeit zu entwikkelnden Lernalgorithmus durchführen zu können: Auf die programmtechnische Realisierung des Systems geht Ring nicht ein. Die verwendeten Lernstrategien sind nur in ihrer Makrostruktur dargestellt. Weiterhin fehlt eine empirisch fundierte Evaluierung

von PELIKAN.

Ob sich PELIKAN für die maschinelle Erlernung und Simulation morphologischer Ableitungsregeln eignet, läßt sich auf der Grundlage von Rings Ausführungen nicht entscheiden.

3 PROBLEMSTELLUNG UND HYPOTHESEN

In diesem Abschnitt werden die Problemstellung und die Hypothesen dieser Arbeit ausführlich dargestellt. Zunächst werden wir in Unterabschnitt 3.1 die Problemstellung ausbreiten und dabei auch den größten Teil der für diese Arbeit spezifischen Termini einführen. In 3.2 werden die Hypothesen der Arbeit formuliert. Abschließend werden wir uns in 3.3 in einem Exkurs mit einigen Fragestellungen befassen, die mit den Ausführungen in 3.1 und 3.2 zusammenhängen: Zum einen soll genauer begründet werden, warum der zu konstruierende Lernalgorithmus nicht Modell sein kann für die Erlernung morphologischer Regeln durch den Menschen. Zum anderen soll der Bezug des Lernalgorithmus zu empiristischen und rationalistischen Spracherwerbstheorien geklärt werden.

3.1 Problemstellung

Es soll ein System zur maschinellen Erlernung und Simulation morphologischer Ableitungsregeln entwickelt und unter linguistischen Fragestellungen evaluiert werden. Unter einer morphologischen Ableitungsregel verstehen wir eine Regel, die angibt, wie eine von 2 Wortformen, zwischen denen eine morphologische Ableitungsbeziehung besteht, aus der anderen Wortform erzeugt wird. Da wir rein synchron vorgehen, ist es dabei für uns irrelevant, welche der beiden Wortformen historisch aus der anderen abgeleitet ist und ob überhaupt eine der beiden Wortform historisch aus der anderen entstanden ist oder ob beide Wortformen auf eine dritte Wortform zurückgehen. Wesentlich ist für uns ausschließlich, daß zwischen den 2 Wortformen eine morphologische Ableitungsbeziehung besteht. Wie der Ausdruck 'morphologische Ableitungsbeziehung' zu verstehen ist, werden wir weiter unten erläutern.

Das zu entwickelnde System soll sich gemäß seinen beiden Funktionen Erlernung und Simulation und dem Gegenstand von Erlernung und Simulation aus 3 Teilkomponenten zusammensetzen (vgl. dazu auch Abb. 1.1):

- Einem *Lernalgorithmus*, der Regeln erlernt,
- einer *Wissensbasis*, in der erlernte Regeln langfristig aufbewahrt werden,
- und einem *Anwendungsalgorithmus*, der - in der Wissensbasis abgelegte - Regeln anwendet.

Als Objektbereich, aus dem er morphologische Ableitungsregeln erlernen soll, soll dem Lernalgorithmus eine Menge geordneter Wortpaare vorgegeben werden. Diese Wortpaarmenge muß nach bestimmten Regeln aufgebaut sein, die wir im folgenden wiedergeben werden. Es handelt sich bei der Wiedergabe dieser Regeln um eine recht trockene Darstellung, die aber unerläßlich ist, da für eine richtige Einschätzung der Leistungen des zu entwickelnden Algorithmus auch eine genaue Spezifikation des erforderlichen Inputs gegeben werden muß. Die Regeln lauten im einzelnen:

I Die 2 Komponenten jedes Wortpaares sollen Zeichenketten sein, die jeweils graphische Realisierungen von Wörtern einer beliebigen - jedoch *innerhalb einer* vorgegebenen Menge von Wortpaaren - festen Schriftsprache sind, welche das lateinische Alphabet verwendet. Die erste, nichtleere Komponente jedes Paares nennen wir *Quellstring*, die zweite, die auch leer sein kann, *Zielstring*. Die gegebene Menge geordneter Paare wird im folgenden *Lernkorpus*, zuweilen auch *Wortpaarliste* genannt.

Leichte Abweichungen vom lateinischen Alphabet, wie beispielsweise Erweiterungen um die diakritischen Zeichen für Akzente im Französischen oder für Umlaute im Deutschen, sollen in den Wortpaarlisten zulässig sein.

II Jeder nicht-leere Zielstring eines geordneten Paares soll in einer morphologischen Ableitungsbeziehung zu dem entsprechenden Quellstring stehen. Der Ausdruck 'morphologische Ableitungsbeziehung' ist dabei folgendermaßen zu verstehen:

Def. *morphologische Ableitungsbeziehung*: Morphologische Ableitungsbeziehungen sind *alle und nur die* zweistelligen Relationen, die zwischen 2 formal ähnlichen Zeichenketten A, B bestehen, welche *semantisch verschiedene, aber ähnliche* Wortformen einer natürlichen Sprache repräsentieren, wobei der Stamm von B aus dem Stamm S von A entsteht

1. durch Konkatenation von S mit einem, ggf. leeren Wortbildungssuffix
2. oder durch Ersetzung eines nicht-leeren Wortbildungssuffixes am Ende von S durch ein anderes, nicht-leeres Wortbildungssuffix
3. oder durch Konkatenation eines nicht-leeren Wortbildungspräfixes mit S
4. oder durch Ersetzung eines nicht-leeren Wortbildungspräfixes am Anfang von S durch ein anderes nicht-leeres Wortbildungspräfix
5. oder durch Konkatenation eines nicht-leeren Wortbildungspräfixes und eines nicht-leeren Wortbildungssuffixes mit S, wobei Wortbildungspräfix und -suffix gemeinsam auftreten müssen.

Bei jedem dieser 5 Fälle *können* - zusätzlich zu der Konkatenation mit, bzw. der Ersetzung von Prä- und Suffixen - weitere graphische Veränderungen von S auftreten.

Diese Definition bedarf einiger Erläuterungen:

- Es ist irrelevant, ob in einer Wortpaarliste der Quell- oder der Zielstring die Größe A, bzw. B der Definition repräsentiert.
- Als *Stamm* einer Zeichenkette C betrachten wir diejenige Zeichenkette, die aus C durch Tilgung des Flexionssuffixes - soweit vorhanden - entsteht. Weist C kein Flexionssuffix auf, so ist der Stamm gleich C. Stamm zu 'treiben' ist demnach 'treib', Stamm zu 'Tag' ist 'Tag'.
- Die bei Fall 1 eingeordnete Ableitung mittels des leeren Wortbildungssuffixes, also des Nullsuffixes, ohne gleichzeitige Veränderung von S wird, falls S = A und B einer anderen Wortart angehört als A, auch als *Konversion* (= "Übertritt von Wörtern in eine andere Wortart ohne formale Änderung" (Lewandowski 1976: 357)) bezeichnet.[1] Dagegen bezeichnet man die Ableitung mittels eines Nullsuffixes bei gleichzeitiger Veränderung von S sowie die Ableitung mittels eines Nullsuffixes, falls S ≠ A, auch als *implizite Ableitung*.[2]

1 Vgl. dazu z. B. Fleischer (1974: 74-76) und Henzen (1947: 249-253).
2 Vgl. Fleischer (1974: 72-74).

- Unter '*weiteren graphischen Veränderungen von S*' verstehen wir in der Definition jegliche von Präfigierung und Suffigierung verschiedene graphische Veränderung des Stamms. Wie auch die untenstehenden Beispiele veranschaulichen, können dies im Deutschen z. B. durch Synkopen, Apokopen, Ablautungen, Umlautungen oder Rückumlautungen bedingte Veränderungen sein, sowie Veränderungen, die durch das Einfügen oder Tilgen von Infixen oder Fugenmorphemen hervorgerufen werden.

Die 5 Fälle der Bildung des Stamms von B aus dem Stamm S von A wollen wir nun mit einigen kommentierten Beispielen aus dem Gegenwartsdeutschen veranschaulichen:

Beispiele zu Fall 1

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
'führen'	'Führung'	S wird mit dem Wortbildungssuffix 'ung' konkateniert.
'liegen'	'Lage'	S wird mit dem Wortbildungssuffix 'e' konkateniert, wobei aufgrund einer Ablautung eine graphische Veränderung von S auftritt ('ie' wird ersetzt durch 'a').
'Tag'	'täglich'	S wird mit dem Wortbildungssuffix 'lich' konkateniert, wobei aufgrund einer Umlautung eine graphische Veränderung von S auftritt ('a' wird ersetzt durch 'ä').
'nähen'	'Naht'	S wird mit dem (heutzutage nicht mehr produktiven) Wortbildungssuffix 't' konkateniert, wobei aufgrund einer Rückumlautung eine graphische Veränderung von S auftritt ('ä' wird er-

Beispiele zu Fall 1 (Fortsetzung)

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
		setzt durch 'a').
'ankommen'	'Ankunft'	S wird mit dem Wortbildungssuffix 't' konkateniert, wobei aufgrund einer Stammkonsonanten- und -vokalveränderung eine graphische Veränderung von S auftritt ('omm' wird ersetzt durch 'unf').
'morgen'	'morgendlich'	S wird mit dem Wortbildungssuffix 'lich' konkateniert, wobei aufgrund der Einfügung eines Fugenmorphems ('d') eine graphische Veränderung von S auftritt.
'fortrennen'	'Fortgerenne'	S wird mit dem Wortbildungssuffix 'e' konkateniert, wobei aufgrund der Einfügung eines Infixes ('ge') eine graphische Veränderung von S auftritt.
'englisch' (= Adjektiv)	'Englisch' (= Substantiv)	S wird mit dem leeren Wortbildungssuffix konkateniert.
'reiten'	'Ritt'	S wird mit dem leeren Wortbildungssuffix konkateniert, wobei aufgrund einer Ablautung eine graphische Veränderung von S auftritt ('eit' wird ersetzt durch 'itt').
'ausdrücken'	'Ausdruck'	S wird mit dem leeren Wortbildungssuffix konkateniert, wobei aufgrund einer Rückumlautung eine graphische Veränderung von S auftritt ('ü'

Beispiele zu Fall 1 (Fortsetzung)

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
		wird ersetzt durch 'u').
'vorziehen'	'Vorzug'	S wird mit dem leeren Wortbildungssuffix konkateniert, wobei aufgrund einer Stammkonsonanten- und -vokalveränderung eine graphische Veränderung von S auftritt ('ieh' wird ersetzt durch 'ug').
'aufhalten'	'Aufenthalt'	S wird mit dem leeren Wortbildungssuffix konkateniert, wobei aufgrund der Einfügung eines Infixes ('ent') eine graphische Veränderung von S auftritt.

Abb. 3.1

Beispiele zu Fall 2

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
'blamieren'	'Blamage'	Das Wortbildungssuffix 'ier' am Ende von S wird durch das Wortbildungssuffix 'age' ersetzt.
'klassifizieren'	'Klassifikation'	Das Wortbildungssuffi 'ier' am Ende von S wird durch das Wortbildungssuffix 'ation ersetzt, wobei aufgrund einer Stammkonsonantenveränderung eine graphische Verän derung von S auftritt ('z' wird ersetzt dur 'k').

Beispiele zu Fall 2 (Fortsetzung)

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
'komprimieren'	'Kompression'	Das Wortbildungssuffix 'ier' am Ende von S wird durch das Wortbildungssuffix 'ion' ersetzt, wobei aufgrund einer Stammkonsonanten- und -vokalveränderung eine graphische Veränderung von S auftritt ('im' wird ersetzt durch 'ess').

Abb. 3.2

Beispiele zu Fall 3

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
'plaudern'	'Geplauder'	Das Wortbildungspräfix 'Ge' wird mit S konkateniert.
'singen'	'Gesang'	Das Wortbildungspräfix 'Ge' wird mit S konkateniert, wobei aufgrund einer Ablautung eine graphische Veränderung von S auftritt ('i' wird ersetzt durch 'a').
'zanken'	'Gezänk'	Das Wortbildungspräfix 'Ge' wird mit S konkateniert, wobei aufgrund einer Umlautung eine graphische Veränderung von S auftritt ('a' wird ersetzt durch 'ä').
'Koks'	'verkoken'	Das Wortbildungspräfix 'ver' wird mit S konkateniert, wobei aufgrund einer Stammkonsonantenveränderung eine graphische Veränderung von S auftritt ('s' wird getilgt).

Abb. 3.3

Beispiele zu Fall 4

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
'Gerümpel'	'entrümpeln'	Das Wortbildungspräfix 'Ge' am Anfang von S wird durch das Wortbildungspräfix 'ent' ersetzt.
'Gemahl'	'vermählen'	Das Wortbildungspräfix 'Ge' am Anfang von S wird durch das Wortbildungspräfix 'ver' ersetzt, wobei aufgrund einer Umlautung eine graphische Veränderung von S auftritt ('a' wird ersetzt durch 'ä').

Abb. 3.4

Beispiele zu Fall 5

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
'Nachricht'	'benachrichtigen'	Das Wortbildungspräfix 'be' und das Wortbildungssuffix 'ig' werden mit S konkateniert.
'treiben'	'Getriebe'	Das Wortbildungspräfix 'Ge' und das Wort bildungssuffix 'e' werden mit S konkateniert, wobei aufgrund einer Ablautung eine graphische Veränderun von S auftritt ('ei' wird ersetzt durch 'ie').
'Vollmacht'	'bevollmächtigen'	Das Wortbildungspräfix 'be' und das Wort bildungssuffix 'ig' werden mit S konkateniert, wobei aufgrund einer Umlautung eine

Beispiele zu Fall 5 (Fortsetzung)

A	B	Kommentierung der Bildung des Stamms von B aus dem Stamm S von A
		graphische Veränderung von S auftritt ('a' wird ersetzt durch 'ä').
'Gnade'	'begnadigen'	Das Wortbildungspräfix 'be' und das Wortbildungssuffix 'ig' werden mit S konkateniert, wobei aufgrund der Apokope des unbetonten Vokals ('e') in der Endsilbe von S eine graphische Veränderung von S auftritt.

Abb. 3.5

Es muß an dieser Stelle nachdrücklich betont werden, daß die Definition des Begriffes 'morphologische Ableitungsbeziehung' hier ausschließlich eingeführt wurde, um den Objektbereich des in der vorliegenden Arbeit zu entwickelnden Lernalgorithmus abzugrenzen. Es wird nicht etwa der Anspruch erhoben, eine für Linguisten in allen Situationen brauchbare Definition vorzulegen.

Die Definition schließt sich in einigen Punkten nicht an existierende Definitionen an (, die ihrerseits allerdings auch sehr heterogen sind): Es werden nämlich einerseits sprachliche Gegebenheiten wie implizite Ableitung (vgl. die letzten 5 Beispiele zu Fall 1) und Präfigierung (vgl. Fall 3-5) einbezogen, die an anderer Stelle nicht unter den Begriff der 'morphologischen Ableitungsbeziehung' subsumiert werden.[3] Andererseits werden in der Definition bestimmte

3 Polenz (1973: 145) und Hoeppner (1980: 10) rechnen die Präfigierung und die implizite Ableitung nicht zur derivativen Wortbildung. Fleischer (1974: 53 u. 63) zählt zwar die implizite Ableitung, jedoch nicht die Präfigierung zur derivativen Wortbildung. Die Duden-Grammatik (1973: 345) subsumiert dagegen unter die derivative Wortbildung sowohl Präfigierung als auch implizite Ableitung.

sprachliche Ausdrucksmittel für Ableitungsbeziehungen vernachlässigt. Hierzu gehören die sogenannten Suprafigierungen. Dies sind Beifügungen suprasegmentaler Morphe, wie Betonungen, die Ableitungsbeziehungen ausdrücken können (Beispiel: engl. /im'pɔ:t/ <einführen> <==> /'impɔ:t/ <Einfuhr>). Suprafigierungen mußten wir ausschließen, da der zu entwickelnde Algorithmus ausschließlich Zeichenketten über dem lateinischen Alphabet bearbeiten soll, das lateinische Alphabet jedoch keine Graphen für suprasegmentale Morphe enthält. Die Ableitungsbeziehung zwischen 'import' <einführen> und 'import' <Einfuhr> wird daher wie eine Konversion behandelt. Neben den Suprafigierungen läßt unsere Definition Ableitungsbeziehungen zwischen Wortgruppen und Einzelwörtern ausser acht, wie etwa bei

'Auskunft geben' <==> 'Auskunftgeber'

und

'instand setzen' <==> 'Instandsetzung'.

III Innerhalb einer Wortpaarliste soll die Bedeutung der Wortbildungsmorpheme, mit denen die Zielstrings aus den Quellstrings entstehen, für alle Zielstrings annähernd gleich sein.

Es muß hier nachdrücklich betont werden, daß der zu konstruierende Lernalgorithmus in der Lage sein soll, auch für solche Wortpaarlisten, die Bedingung III nicht erfüllen, die zwischen den Quell- und Zielstrings bestehende Ableitungsbeziehung zu erlernen. Fraglich ist jedoch, welchem übergeordneten Zweck, sei es bei wissenschaftsinternen, sei es bei wissenschaftsexternen Anwendungen, eine Erlernung solcher Ableitungsbeziehungen dienen kann.

Die Bedingung soll hier mit einem Beispiel veranschaulicht werden: Es sei eine Wortpaarliste gegeben, deren Quellstrings Verben der deutschen Gegenwartssprache sind. Die Zielstrings dieser Liste seien ebenfalls Verben, die gegenüber den Quellstrings die zusätzliche, durch Wortbildungsmorpheme getragene Bedeutung <resultativ> haben, d. h., sie bezeichnen die Vollendung des durch die Quellstrings

ausgedrückten Geschehens.[4] Der Wortpaarliste können also u. a. folgende Paare angehören:

('frieren', 'erfrieren'),
('blühen', 'verblühen'),
('klingen', 'verklingen').

Sie darf jedoch nicht die folgenden Paare enthalten, obwohl das darin verwendete Präfixmorphem graphisch identisch ist mit einem der Präfixmorpheme in den eben angeführten Paaren:

('fahren', 'verfahren'),
('schreiben', 'verschreiben').

Das Präfix 'ver' in 'verfahren' und 'verschreiben' trägt nämlich nicht die Bedeutung <resultativ>, sondern es verleiht dem Verb die "Bedeutungsmodifikation 'in die Irre', 'irrtümlich', 'falsch'." (Kürschner 1975b: 363)

IV In manchen Fällen existieren Quellstrings, die zwar derselben Gruppe von Wörtern angehören wie die in einer Wortpaarliste mit (nicht-leerem) Zielstring vorhandenen Quellstrings, jedoch gibt es zu diesen Quellstrings keinen Zielstring, der zu ihnen in einer morphologischen Ableitungsbeziehung steht, die dieselbe semantische Beziehung ausdrückt, die durch die morphologische Ableitungsbeziehung in den Wortpaaren mit nicht-leerem Zielstring repräsentiert wird. Solche Quellstrings sind *ohne* Zielstring in die Wortpaarliste aufzunehmen.

Beispiel: Sind die Quellstrings einer Wortpaarliste französische Adjektive in ihrer femininen Form und sind die Zielstrings die aus diesen Adjektiven abgeleiteten Adverbien, so ist 'bonne' <gute> ein Quellstring, zu dem kein Zielstring existiert, denn es gibt kein entsprechendes Adverb, das zu 'bonne' in einer morphologischen Ableitungsbeziehung steht (Semantisch betrachtet ist zwar 'bien' Abverb zu 'bonne', jedoch nicht aus morphologischer Sicht). Das Adjektiv 'bonne' ist demnach ohne Zielstring in die Wortpaarliste aufzunehmen.

V In einer Wortpaarliste dürfen keine Paare mit identischem Quell-, aber verschiedenem Zielstring auftreten. Die Zuordnung der Ziel- zu den Quellstrings muß also *rechtseindeutig* sein.

Forderung V wurde lediglich eingeführt, um den zu konstruie-

4 Vgl. Duden 1973: 65.

renden Algorithmus zunächst nicht zu komplex werden zu lassen. Langfristig ist es sicher sinnvoll, wenn die von Forderung V vorgeschriebene Rechtseindeutigkeit von Wortpaarlisten zugunsten von mehrmehrdeutigen Wortpaarlisten aufgegeben wird. Nach Forderung V dürfen beispielsweise in einer Wortpaarliste, deren Quellstrings Verben der deutschen Gegenwartssprache sind und deren Zielstrings die entsprechenden Nomina actionis sind, nicht gleichzeitig die 2 Paare

('übertreten', 'Übertritt')

und

('übertreten', 'Übertretung')

auftreten. Dagegen dürfen durchaus die 2 Paare

('ausdrücken', 'Ausdruck')

und

('ausdrucken', 'Ausdruck')

in derselben Wortpaarliste auftreten, da dadurch die Rechtseindeutigkeit der Liste nicht beeinträchtigt wird.

Zur Veranschaulichung der vorangehenden Konventionen für den Aufbau von Wortpaarlisten geben wir im folgenden einige Beispiele für zulässige und unzulässige Wortpaarlisten:

Zulässige Wortpaarlisten

- Quellstring = frz. Verb,
  Zielstring = frz. Nomen actionis:

('acclamer'	<zujubeln>,	'acclamation'	<Beifallsruf>)
('affirmer'	<behaupten>,	'affirmation'	<Behauptung>)
('balbutier'	<hervorstammeln>,	'balbutiement'	<Stammelei>)
('diriger'	<leiten>,	'direction'	<Leitung>)
('discuter'	<erörtern>,	'discussion'	<Erörterung>)
('fumer'	<rauchen>,	'')	
('humilier'	<demütigen>,	'humiliation'	<Demütigung>)

- Quellstring = frz. Nomen agentis,
  Zielstring = frz. Nomen actionis:

('directeur'	<Leiter>,	'direction'	<Leitung>)
('édificateur'	<Erbauer>,	'édification'	<Erbauung>)
('facteur'	<Orgelbauer>,	'facture'	<Orgelbau>)
('immigrant'	<Einwanderer>,	'immigration'	<Einwanderung>)
('instructeur'	<Ausbilder>,	'instruction'	<Ausbildung>)

Unzulässige Wortpaarlisten

- Quellstring = nhd. Adverb im Positiv,
  Zielstring = nhd. Adverb im Komparativ:

  ('alt', 'älter')
  ('gut', 'besser')
  ('schnell', 'schneller')
  ('schön', 'schöner')

  Vorliegender Verstoß gegen die Regeln für den Aufbau von Wortpaarlisten: Zwischen Adverbien im Positiv und Adverbien im Komparativ besteht keine morphologische Ableitungsbeziehung (vgl. Regel II). Sie sind lediglich verschiedene Flexionsformen derselben Grundform.

- Quellstring = nhd. Verb,
  Zielstring = nhd. Nomen actionis:

  ('arbeiten', 'Arbeit')
  ('', 'Dienstenthebung')
  ('erneuern', 'Erneuerung')
  ('stationieren', 'Station')
  ('visitieren', 'Visitation')
  ('visitieren', 'Visite')

  Vorliegende Verstöße gegen die Regeln für den Aufbau von Wortpaarlisten:
  - Der Quellstring des Paares ('', 'Dienstenthebung') ist leer. Quellstrings dürfen jedoch nicht leer sein, sondern nur Zielstrings. (Vgl. Regel I.)
  - Die Bedeutung der Wortbildungsmorpheme, mit denen die Zielstrings aus den Quellstrings entstehen, ist nicht für alle Zielstrings gleich: Der Zielstring 'Station' ist ein Nomen loci, während alle anderen Zielstrings Nomina actionis sind. (Vgl. Regel III.)
  - Die Zuordnung der Quell- zu den Zielstrings ist nicht rechtseindeutig, da die Wortpaarliste die 2 Paare ('visitieren', 'Visitation') und ('visitieren', 'Visite') enthält. (Vgl. Regel V.)

Zu jeder Wortpaarliste, die den oben eingeführten Regeln entspricht, soll der zu entwickelnde Lernalgorithmus eine Menge von Regeln konstruieren können, die bei späterer maschineller Anwendung auf die Quellstrings Zeichenketten erzeugt, die mit den in der Wortpaarliste vorgegebenen Zielstrings identisch sind. War in der Wortpaarliste kein Zielstring vorgegeben, so ist dementsprechend kein Zielstring zu erzeugen. Unter der Voraussetzung, daß die jeweils vorliegende morphologische Ableitungsbeziehung auch regelhaft ist, soll die Regelmenge weiterhin die Eigenschaft haben, daß sie auch korrekte Zielstrings erzeugt, wenn sie auf solche Quellstrings angewendet wird, die nicht in der

ursprünglich vorgegebenen Wortpaarliste enthalten waren, die jedoch derselben Gruppe von Wörtern angehören wie die in der Wortpaarliste enthaltenen Quellstrings. Diese Eigenschaft der Regelmenge nennen wir im folgenden *prädiktive Kapazität*. Wenn beispielsweise in der Lernphase die 3 Wortpaare ('führen', 'Führung'), ('behandeln', 'Behandlung') und ('produzieren', 'Produktion') vorgegeben waren, so sollen in der Anwendungsphase nicht nur zu 'führen', 'behandeln' und 'produzieren' die korrekten Zielstrings erzeugt werden, sondern es soll z. B. auch 'Handlung' zu 'handeln' und 'Induktion' zu 'induzieren' gebildet werden.

Im folgenden wollen wir erörtern, welche allgemeine Form die vom Lernalgorithmus zu erlernende Regelmenge haben soll, m. a. W. in welcher *formalen Beschreibungssprache* die Regelmenge dargestellt werden soll.

Die morphologische Ableitungsbeziehung, die zwischen Quell- und Zielstring besteht, drückt sich auf der Ausdrucksebene in der Regel durch graphische Unterschiede aus[5], wie etwa bei folgendem Beispiel:

'produzieren' <==> 'produktion'

Eine Möglichkeit, den Ziel- aus dem Quellstring zu erzeugen, ist die Verwendung von Regeln, die ausschließlich auf die graphische Gestalt von Quell- und Zielstring Bezug nehmen und die vorhandenen graphischen Unterschiede austauschen. Im obigen Beispiel würde also eine solche Regel den Zielstring 'produktion' aus dem Quellstring 'produzieren' erzeugen, indem sie im Quellstring 'zieren' durch 'ktion' ersetzt.

Die Verwendung von Regeln, die sich an der Graphie orientieren, ist durchaus nicht so abwegig, wie es zunächst vielleicht erscheinen mag. Psycholinguistische Untersuchungen[6] zur Erler-

5 Eine Ausnahme bildet die Konversion, bei der ja Quell- und Zielstring identisch sind.

6 Es handelt sich hierbei insbesondere um die Arbeiten von Berko (1958), Derwing (1976), Derwing/Baker (1977) und Derwing/Baker (1979).

nung derivations- und flexionsmorphologischer Regeln haben gezeigt,

- daß tatsächlich *Regeln* erlernt werden, die auch eine *prädiktive Kapazität* besitzen, und nicht etwa lediglich Wortformen memoriert werden
- und daß diese Regeln auch sehr häufig über der *Lautgestalt* operieren, die sich bei Sprachen mit Alphabetschrift ja verhältnismäßig eindeutig in der Graphie widerspiegelt.

So führen etwa Bruce L. Derwing und William J. Baker zur *Erlernung von Regeln mit prädiktiven Kapazität* aus:

> "Rather than learning specific instances of language, it is fairly evident that the child learns generalizations about the language system, generalizations he can then apply in relatively novel situations which are different from his specific learning experiences. He can produce and understand, as a result of these generalizations, specific utterances he has never encountered before." (Derwing/Baker 1977: 86)

Zur *Orientierung morphologischer Regeln an Oberflächenphänomenen*, wie etwa der Lautstruktur, stellen sie für das Englische *allgemein* fest:

> "Both in the areas of inflectional and derivational morphology, our results confirm the view that a variety of kinds of *surface regularities* (...; Auslassung des Verf.) are extracted by language learners and productively employed by them in producing novel speech forms. (Hervorhebung des Verf.)" (Derwing/Baker 1977: 108)

Speziell im Hinblick auf die Pluralbildung englischer Substantive führen sie konkreter aus:

> "These findings leave us with the conclusion (...; Auslassung des Verf.) that the rules that are learned at every stage seem on the evidence to take some notion of the subsegmental phonetic feature into account. Our data also continue to permit the view that the rules involved might be of a very general phonotactic nature, and hence operative across more than one morphological category." (Derwing/Baker 1977: 100)

Auch aus computerlinguistischer Sicht gibt es Argumente, die für die Verwendung graphisch orientierter Regeln im Bereich der Derivationsmorphologie sprechen. So arbeiten beispielsweise Pacak und Pratt mit graphisch basierten Regeln (vgl. Abschnitt 2.1.3). Wolfgang Hoeppner konnte in seiner Dissertation "Derivative Wortbildung der deutschen Gegenwartssprache und ihre algo-

rithmische Analyse" für zahlreiche Zeichenketten, die potentiell Wortbildungssuffixe des Deutschen sind, graphische Kontextbedingungen formulieren, die erfüllt sein müssen, damit unter den *potentiellen* Wortbildungssuffixen maschinell auch die *tatsächlichen* Suffixe ausgemacht werden können. Er schreibt dazu u. a.:

> "Die wichtigste Beschreibungsebene, auf der Restriktionen für einzelne Suffixe formuliert werden können, ist die graphemische." (Hoeppner 1980: 71)[7]

Für den zu konstruierenden Lernalgorithmus sind verschiedene Formen graphisch basierter Regeln denkbar. Wir verwenden im folgenden 4 verschiedene Typen. Der erste Typ, den wir 'präfixorientierte Substitutionsanweisung' nennen, dient zur Durchführung graphischer Veränderungen am Wortanfang. Solche Veränderungen werden insbesondere dann notwendig, wenn sich Quell- und Zielstring infolge einer Präfigierung unterscheiden. Mit einer präfixorientierten Substitutionsanweisung läßt sich beispielsweise 'Abgesang' aus 'Gesang' erzeugen.

> Def. *präfixorientierte Substitutionsanweisung*: Präfixorientierte Substitutionsanweisungen sind alle und nur die Anweisungen der Form
>
> $X \rightarrow Y/\#__ (Z_1 | \ldots |Z_i| \ldots |Z_n)$, wobei
>
> - X und Y - ggf. leere - Zeichenketten sind,
> - $Z_1, \ldots, Z_n$ nicht-leere Zeichenketten sind,
> - $n >= 0$.

Diese Anweisung ist folgendermaßen zu lesen: Ersetze X durch Y, falls X am Anfang eines Quellstrings steht, und zwar unmittelbar links von Z_1 oder ... oder Z_i oder ... oder Z_n.

In der Anweisung markiert '#' die Wortgrenze. '__' gibt die Position an, auf der X im Wort auftreten muß, um durch Y ersetzt werden zu können, nämlich am Wortanfang und unmittelbar links von Z_1 oder ... oder Z_i oder ... oder Z_n.

Der zweite Anweisungstyp dient zur Durchführung graphischer Veränderungen am Wortende, wie sie etwa notwendig sind, um 'erforderlich' aus 'erfordern' zu erzeugen. Da solche Veränderungen

---

7 Hoeppner verwendet den Ausdruck 'graphemisch' in demselben Sinn, wie wir in der vorliegenden Arbeit den Ausdruck 'graphisch' verwenden.

meist dann vorzunehmen sind, wenn sich Quell- und Zielstring aufgrund einer Suffigierung unterscheiden, nennen wir diesen zweiten Anweisungstyp '*suffix*orientierte Substitutionsanweisung'.

Def. *suffixorientierte Substitutionsanweisung*: Suffixorientierte Substitutionsanweisungen sind alle und nur die Anweisungen der Form

$X \rightarrow Y/(Z_1| \ldots |Z_i| \ldots |Z_n)\ __\ \#$, wobei

- X und Y - ggf. leere - Zeichenketten sind,
- $Z_1, \ldots, Z_n$ nicht-leere Zeichenketten sind,
- $n >= 0$.

Diese Anweisung ist folgendermaßen zu lesen: Ersetze X durch Y, falls X am Ende eines Quellstrings steht, und zwar unmittelbar rechts von Z_1 oder ... oder Z_i oder ... oder Z_n.

Die präfix- und die suffixorientierten Substitutionsanweisungen sind kontext*sensitive* Regeln, falls $n > 0$, da X dann nicht in beliebigen Kontexten durch Y ersetzt werden kann, sondern nur, wenn es unmittelbar links (bei präfixorientierten Anweisungen) oder unmittelbar rechts (bei suffixorientierten Anweisungen) von Z_1 oder ... oder Z_n steht. Für $n = 0$ ist eine Substitutionsanweisung kontext*frei*.

X nennen wir im folgenden *Substituendum*, Y nennen wir *Substituens* und für jedes i, mit $1 <= i <= n$, nennen wir Z_i *Kontextstring*. Der rechts von '/' stehende Ausdruck '$\#\ __\ (Z_1| \ldots |Z_i| \ldots |Z_n)$', bzw. '$(Z_1| \ldots |Z_i| \ldots |Z_n)\ __\ \#$' wird *Kontextbedingung* genannt, falls $n > 0$.

Die Erzeugbarkeit von Zielstrings aus Quellstrings mittels Substitutionsanweisungen soll nun für suffixorientierte Substitutionsanweisungen an einem Beispiel veranschaulicht werden. Um den Zweck der Veranschaulichung auch erfüllen zu können, wollen wir das Beispiel nicht zu komplex machen und ignorieren daher bewußt einige Übergeneralisierungen und Überdiskriminierungen bei der Konstruktion der Anweisungen: Die Substitutionsanweisung, die das französische Adverb 'clairement' <klar> aus dem femininen Adjektiv 'claire' erzeugt, lautet

'' → 'ment'/__ #.

Diese Anweisung ist kontextfrei. Sie erzeugt auch 'rarement'

<selten> aus 'rare'. Dagegen ist sie unbrauchbar für die Paare

('prudente' <vorsichtig>, 'prudemment'),
('jolie' <hübsch>, 'joliment'),
('mauvaise' <schlecht>, ''),

da ihre Anwendung auf 'prudente' nicht etwa 'prudemment' ergibt, sondern *'prudentement', und da die Anwendung auf 'jolie' nicht 'joliment' ergibt, sondern *'joliement'. Weiterhin würde aus 'mauvaise' nicht '' erzeugt, sondern *'mauvaisement'.[8] Um aus 'prudente' das Adverb 'prudemment' zu gewinnen, ist die Anweisung

'nte' → 'mment'/__#

geeignet. 'joliment' läßt sich aus 'jolie' mit der Anweisung

'e' → 'ment'/__#

erzeugen. Um die Bildung eines Adverbs zu 'mauvaise' zu verhindern, verwenden wir die Anweisung

'se'# ¬,

die Realisierung eines neuen Anweisungstyps ist:

Def. *suffixorientierte Blockierungsanweisung*: Suffixorientierte Blockierungsanweisungen sind alle und nur die Anweisungen der Form

X# ¬,

wobei X eine - ggf. leere - Zeichenkette ist.

Die Anweisung ist folgendermaßen zu lesen: Endet ein Quellstring auf die Zeichenkette X, so ist zu dem Quellstring kein Zielstring zu bilden (, denn es existiert kein Zielstring, der zu dem Quellstring in der zu beschreibenden morphologischen Ableitungsbeziehung steht).

Neben den *suffix*orientierten Blockierungsanweisungen führen wir noch *präfix*orientierte Blockierungsanweisungen ein:

8 Semantisch betrachtet, ist zwar 'mal' Adverb zu 'mauvaise', jedoch nicht aus morphologischer Sicht: 'mal' steht nicht in einer morphologischen Ableitungsbeziehung zu 'mauvaise'. Ein Adverb zu 'mauvaise', das auch in einer morphologischen Ableitungsbeziehung zu 'mauvaise' steht, existiert nicht.

Def. *präfixorientierte Blockierungsanweisung*: Präfixorientierte Blockierungsanweisungen sind alle und nur die Anweisungen der Form

X ¬,

wobei X eine - ggf. leere - Zeichenkette ist.

Die Anweisung ist folgendermaßen zu lesen: Beginnt ein Quellstring mit der Zeichenkette X, so ist zu diesem Quellstring kein Zielstring zu bilden (, denn es existiert kein Zielstring, der zu dem Quellstring in der zu beschreibenden morphologischen Ableitungsbeziehung steht).

Die Zeichenkette X von Blockierungsanweisungen nennen wir im folgenden *Impediens*.

Blockierungs- und Substitutionsanweisungen (und nur diese) fassen wir unter dem Oberbegriff *Derivationsanweisungen* zusammen. Einen Überblick über die eingeführten Anweisungstypen gibt Abb. 3.6.

Nun zurück zu unserem obigen Beispiel: Um zu jedem Quellstring der Wortpaarliste

('claire', 'clairement'),
('rare', 'rarement'),
('prudente', 'prudemment'),
('jolie', 'joliment'),
('mauvaise', '')

den entsprechenden Zielstring erzeugen zu können, hatten wir die Anweisungen

'' → 'ment'/__#,
'nte' → 'mment'/__#,
'e' → 'ment'/__# und
'se'# ¬

konstruiert. Diese Anweisungen können wir nicht ohne weiteres in *einer* Anweisungsmenge zusammenfassen, die zu jedem Quellstring der Wortpaarliste den korrekten Zielstring erzeugt. Denn '' → 'ment'/__# ist neben 'claire' und 'rare' auch auf 'prudente', 'jolie' und 'mauvaise' anwendbar, wobei sich die unzulässigen Zielstrings *'prudentement', *'joliement' und *'mauvaisement' ergeben. Außerdem ist die Anweisung 'e' → 'ment'/__# nicht nur

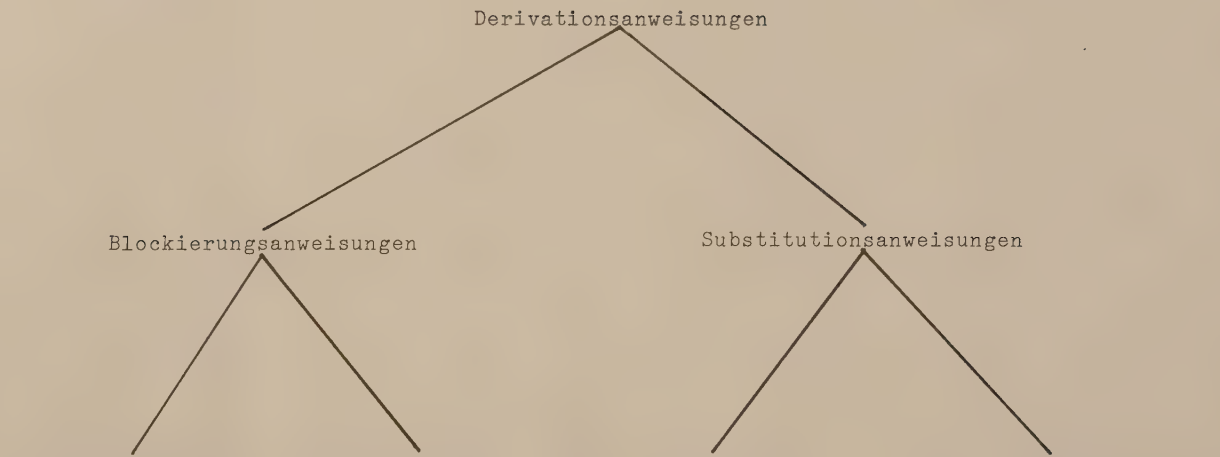

Derivationsanweisungen
Blockierungsanweisungen
Substitutionsanweisungen
Präfixorientierte Blockierungsanweisungen (allg. Form: # X ¬)
Suffixorientierte Blockierungsanweisungen (allg. Form: X # ¬)
Präfixorientierte Substitutionsanweisungen (allg. Form: X → Y/#__ (Z_1|...|Z_n))
Suffixorientierte Substitutionsanweisungen (allg. Form: X → Y/(Z_1|...|Z_n)__ #)

Abb. 3.6

auf 'jolie' anwendbar, sondern auch auf alle übrigen Quellstrings der Wortpaarliste. Dabei ergeben sich die inkorrekten Zielstrings *'clairment', *'rarment', *'prudentment' und *'mauvaisment'. Um dies zu verhindern, müssen wir dafür sorgen, daß keine 2 suffixorientierten Substitutionsanweisungen auf denselben Quellstring angewendet werden können. Eine Möglichkeit, dies zu erreichen, besteht in der Verwendung kontext*sensitiver* Substitutionsanweisungen. Eine entsprechende Anweisungsmenge zum obigen Beispiel ist:

M = {'se'# → ,
'e' → 'ment'/'i'__#,
'' → 'ment'/'re'__#,
'nte' → 'mment'/__#}.

In der obigen Menge M können wir bei Berücksichtigung einer bestimmten Zusatzbedingung eine sogenannte Kürzung vornehmen.

Def. *Kürzung*: Als Kürzung einer Menge M von Derivationsanweisungen werden alle und nur die Veränderungen von M bezeichnet, die bewirken, daß die Gesamtlänge objektsprachlicher Zeichenketten in der aus der Kürzung resultierenden Menge M' kürzer ist als die Gesamtlänge objektsprachlicher Zeichenketten in M.

Die Zusatzbedingung besteht darin, daß die durch die Kürzung entstehende Anweisungsmenge partiell geordnet wird, d. h., daß die Anweisungen nur in einer bestimmten Reihenfolge anzuwenden versucht werden dürfen. Wir führen folgende *partielle Ordnung* ein: Gegeben sei eine Menge

$D = \{pb_1, \ldots, pb_k, sb_1, \ldots, sb_l, ps_1, \ldots, ps_m, ss_1, \ldots, ss_n\}$

von

- präfixorientierten Blockierungsanweisungen $pb_1, \ldots, pb_k$,
- suffixorientierten Blockierungsanweisungen $sb_1, \ldots, sb_l$,
- präfixorientierten Substitutionsanweisungen $ps_1, \ldots, ps_m$,
- und suffixorientierten Substitutionsanweisungen $ss_1, \ldots, ss_n$.

Weiterhin sei ein Quellstring gegeben, zu dem mittels dieser Menge von Anweisungen ein Zielstring erzeugt werden soll. Dann müssen zunächst die Blockierungsanweisungen anzuwenden versucht werden (untereinander können sie in beliebiger Reihenfolge ange-

wendet werden). Ist eine dieser Anwendungen anwendbar, so wird kein Zielstring generiert und es wird auch nicht weiter versucht, irgendeine der anderen Anweisungen anzuwenden. Ist dagegen *keine* der Blockierungsanweisungen anwendbar, so muß die *erste* anwendbare präfixorientierte und die *erste* anwendbare suffixorientierte Substitutionsanweisung bestimmt werden und beide müssen *gleichzeitig* auf den Quellstring angewendet werden, wodurch der Zielstring erzeugt wird.

Um diese Regelung auch in der Notation von Mengen von Derivationsanweisungen zu verdeutlichen, notieren wir - wie in der obigen Menge *D* - im folgenden immer Blockierungs- *vor* Substitutionsanweisungen und - innerhalb dieser beiden Gruppen - immer präfixorientierte *vor* suffixorientierten Anweisungen.

Def. *geordnete Menge von Derivationsanweisungen*: Alle und nur diejenigen Mengen von Derivationsanweisungen
- deren Blockierungsanweisungen vor den Substitutionsanweisungen anzuwenden versucht werden müssen
- *und* deren präfix- und suffixorientierte Substitutionsanweisungen jeweils nur in der Reihenfolge ihres Auftretens in der Menge anzuwenden versucht werden dürfen,

nennen wir geordnete Mengen von Derivationsanweisungen. Falls eine Blockierungsanweisung auf einen Quellstring anwendbar ist, werden die Substitutionsanweisungen nicht mehr anzuwenden versucht. Andernfalls müssen die erste anwendbare präfixorientierte und die erste anwendbare suffixorientierte Substitutionsanweisung bestimmt und gleichzeitig auf den Quellstring angewendet werden.

Alle und nur diejenigen Mengen von Derivationsanweisungen, für die diese Beschränkung hinsichtlich der Anwendungsreihenfolge nicht gilt, nennen wir *ungeordnete Mengen von Derivationsanweisungen*.

Vielleicht ist bei der dargestellten Strategie zur Anwendung von Derivationsanweisungen nicht ohne weiteres verständlich, daß die Möglichkeit vorgesehen ist, auf einen Quellstring sowohl eine präfix- als auch eine suffixorientierte Substitutionsanweisung anzuwenden, um dadurch den entsprechenden Zielstring zu erzeugen. Die Notwendigkeit einer solchen Möglichkeit dürfte jedoch offensichtlich werden, wenn man sich vergegenwärtigt, daß die Derivationsanweisungen auch in der Lage sein sollen, 'benachrichtigen' aus 'Nachricht' zu erzeugen oder 'begnadigen'

aus 'Gnade' (vgl. dazu Fall 5 der Def. morphologische Ableitungsbeziehung). In diesen Fällen müssen graphische Veränderungen sowohl am Anfang als auch am Ende des Quellstrings vorgenommen werden, um den entsprechenden Zielstring zu erzeugen.

Damit *jede* Substitutionsanweisung einer gekürzten geordneten Menge *potentiell* zur Anwendung kommen kann, muß bei der Konstruktion einer solchen Menge folgendes beachtet werden:

1. Für jede präfixorientierte Substitutionsanweisung a, deren Konkatenation von Substituendum und Kontextstring linker Teilstring der Konkatenation von Substituendum und Kontextstring einer anderen präfixorientierten Substitutionsanweisung b ist, darf der Anwendungsversuch erst stattfinden, nachdem festgestellt ist, daß b nicht anwendbar ist.
2. Für jede suffixorientierte Substitutionsanweisung a, deren Konkatenation von Kontextstring und Substituendum rechter Teilstring der Konkatenation von Kontextstring und Substituendum einer anderen suffixorientierten Substitutionsanweisung b ist, darf der Anwendungsversuch erst stattfinden, nachdem festgestellt worden ist, daß b nicht anwendbar ist.

Hinreichende formale Voraussetzung für 1 und 2 ist, daß Anweisung a in der Anweisungsmenge hinter Anweisung b steht. (a muß dabei nicht unmittelbar hinter b stehen.)

Für Blockierungsanweisungen muß keine solche Regelung eingeführt werden, die garantiert, daß jede Blockierungsanweisung potentiell zur Anwendung kommen kann. Wie wir in Abschnitt 5.4 noch sehen werden, werden die Blockierungsanweisungen innerhalb einer Menge von Derivationsanweisungen nämlich so konstruiert, daß keine 2 präfix- bzw. suffixorientierten Blockierungsanweisungen in der Menge existieren, deren Impedienten linke, bzw. rechte Teilstrings voneinander sind. Aus diesem Grund können die Blockierungsanweisungen untereinander auch in beliebiger Reihenfolge anzuwenden versucht werden.

Wir können nun die obige Anweisungsmenge M kürzen zu der Menge

M' = {'se'→,
'e' → 'ment'/'i'__#,
'nte' → 'mment'/__#,
'' → 'ment'/__#},

ohne daß ein Informationsverlust in der Beschreibung der im Lernkorpus gegebenen morphologischen Ableitungsbeziehung eintritt. Dabei legen wir für M' fest, daß es eine geordnete Menge ist. Bei der Kürzung wurde der Kontextstring 're' getilgt. Weiterhin mußten die beiden letzten Substitutionsanweisungen in der Menge vertauscht werden, da sonst nicht gewährleistet gewesen wäre, daß jede Substitutionsanweisung potentiell zur Anwendung kommen kann (s. o.).

Aus der Kürzung ohne Informationsverlust ergibt sich der im folgenden noch wichtige Begriff der Redundanz:

Def. *redundant*: Wir nennen eine Menge von Derivationsanweisungen, die eine morphologische Ableitungsbeziehung beschreibt, redundant gdw. eine Kürzung dieser Menge - ggf. bei gleichzeitigem Vertauschen von Derivationsanweisungen - möglich ist, ohne daß Informationsverlust bezüglich der Darstellung der (im Lernkorpus gegebenen) Ableitungsbeziehung eintritt.

Redundant ist beispielweise die obige Menge M, da sie sich ohne Informationsverlust zu der Menge M' kürzen läßt. Dagegen ist die Menge M' redundanzfrei, da sie sich nicht ohne Informationsverlust kürzen läßt.

Def. *redundanzfrei*: Wir nennen eine Menge von Derivationsanweisungen redundanzfrei gdw. sie nicht redundant ist.

3.2 Hypothesen

Die Hypothesen dieser Arbeit formulieren wir als Forderungen. Wir stellen 5 Forderungen an den zu konstruierenden Algorithmus und die von ihm zu erzeugenden Mengen von Derivationsanweisungen. Die Überprüfung der Forderungen wird Gegenstand von Abschnitt 7 sein.

1. Forderung: Der Lernalgorithmus soll zu jedem Lernkorpus, das den Regeln I-V entspricht, eine Menge maschinell ausführbarer Derivationsanweisungen konstruieren, die bei ihrer Ausführung durch den Anwendungsalgorithmus zu jedem Quellstring des Lernkorpus eine Zeichenkette erzeugt, die identisch ist

mit dem Zielstring, welcher dem Quellstring im Lernkorpus beigegeben ist.

Mengen von Derivationsanweisungen, die diese *Minimal*forderung erfüllen, nennen wir *schwach adäquat*.

2. Forderung: Für jede auf graphischer Ebene zumindest teilweise regelhafte morphologische Ableitungsbeziehung soll gelten: Für jedes zu dieser Ableitungsbeziehung aus linguistischer Sicht *zufällig* zusammengestellte, den Regeln I-V entsprechende Lernkorpus soll der Lernalgorithmus eine Menge von Derivationsanweisungen liefern,
- die entweder, je größer das Lernkorpus ist, im Mittel zu um so mehr an der Ableitungsbeziehung beteiligten und nicht im Lernkorpus enthaltenen Quellstrings den korrekten Zielstring erzeugt
- oder die zu jedem an der Ableitungsbeziehung beteiligten und nicht im Lernkorpus enthaltenen Quellstring den korrekten Zielstring erzeugt.

Die Aufstellung dieser 2. Forderung ist insbesondere durch mögliche Anwendungen des zu entwickelnden Algorithmus motiviert:
- Zum einen ist die Erfüllung der 2. Forderung wichtig, um die Brauchbarkeit des Algorithmus für linguistisch nicht Vorgebildete bei der automatischen Erstellung linguistischer Algorithmen beurteilen zu können.
- Zum anderen ist ihre Erfüllung ausschlaggebender Indikator für die Eignung des Algorithmus für Linguisten, die zu einer morphologischen Ableitungsbeziehung, deren Grad an Regelhaftigkeit *nicht* bekannt ist, Algorithmen erzeugen lassen wollen.

In beiden Anwendungsfällen kann der Benutzer Lernkorpora - linguistisch betrachtet - nur mehr oder weniger *zufällig* zusammenstellen.

3. Forderung: Für jedes zu einer morphologischen Ableitungsbeziehung linguistisch annähernd *repräsentativ* zusammengestellte, den Regeln I-V entsprechende Lernkorpus soll der Lernalgorithmus eine Menge maschinell ausführbarer Derivationsanweisungen liefern, die bei ihrer Ausführung durch den Anwendungsalgorithmus zu mindestens 90% der an der Ableitungsbeziehung beteiligten und nicht im Lernkorpus enthaltenen Quellstrings jeweils den entsprechenden Zielstring erzeugt.

Mengen von Derivationsanweisungen, die diese 3. Forderung erfüllen, nennen wir *stark adäquat*. Eine stark adäquate Menge von Derivationsanweisungen besitzt i. U. zu einer nur schwach adäquaten Menge die oben geforderte Eigenschaft der *prädiktiven Kapazität*

(vgl. Abschnitt 3.1).

Von der Erfüllung der 3. Forderung hängt es ab, inwieweit der zu konstruierende Algorithmus von Linguisten verwendet werden kann, um zu morphologischen Ableitungsbeziehungen, deren Grad an Regelhaftigkeit bekannt ist, effiziente Derivationsalgorithmen erzeugen zu lassen. Nur zu Ableitungsbeziehungen, deren Regelhaftigkeit bekannt ist, lassen sich gezielt *repräsentative* Lernkorpora erstellen. Das Attribut 'repräsentativ' ist folgendermaßen zu verstehen:

> Def. *repräsentatives Lernkorpus*: Wir nennen ein Lernkorpus für eine morphologische Ableitungsbeziehung repräsentativ gdw.
> - die an der Ableitungsbeziehung beteiligten Substitutionsmuster sowie die Kontextbedingungen für die Anwendbarkeit der Substitutionsmuster vollständig in dem Lernkorpus implizit vertreten sind
> - *und* die Blockierungsmuster, d. h. die Impedienten, vollständig in dem Lernkorpus implizit vertreten sind
> - *und* die Häufigkeitsrangfolge, in der die Substitutionsmuster in der gesamten Menge der an der Ableitungsbeziehung in der Sprache beteiligten Wortpaare vorliegen, auch im Lernkorpus gegeben ist.

Der in der Definition auftretende Ausdruck 'Häufigkeitsrangfolge' bedarf einer kurzen Erläuterung: Angenommen, in der gesamten Menge (= Grundgesamtheit) der an einer Ableitungsbeziehung in einer Sprache beteiligten Wortpaare tritt das Substitutionsmuster

$X_1 \rightarrow Y_1$ 50 mal

und das Substitutionsmuster

$X_2 \rightarrow Y_2$ 70 mal auf.

$X_2 \rightarrow Y_2$ tritt also *häufiger als* $X_1 \rightarrow Y_1$ auf. Um nun die Häufigkeitsrangfolge der beiden Substitutionsmuster in der Grundgesamtheit auch für das Lernkorpus gelten zu lassen, genügt es, daß sie im Lernkorpus genauso die 'häufiger als'-Relation erfüllen wie in der Grundgesamtheit. Es genügt also, wenn im Lernkorpus

$X_1 \rightarrow Y_1$ 1 mal

und

$X_2 \rightarrow Y_2$ 2 mal auftritt.

Vielleicht fragt man sich, warum überhaupt die Häufigkeitsrangfolge der Substitutionsmuster in einem repräsentativen Lernkorpus berücksichtigt werden muß: Der Lernalgorithmus muß aufgrund des Lernkorpus Voraussagen darüber machen können, welche Substitutionsmuster in der Grundgesamtheit eher den Status von Regeln und welche eher den Status von Ausnahmen haben, um dadurch bei der Konstruktion der Substitutionsanweisungen Übergeneralisierungen und Überdiskriminierungen vermeiden zu können. Zu den regelhafteren, also den häufigeren Substitutionsmustern muß er nämlich allgemeinere Substitutionsanweisungen konstruieren, zu den weniger regelhaften, also selteneren Mustern muß er speziellere Anweisungen konstruieren. Um dem Lernalgorithmus nun überhaupt zu ermöglichen, aufgrund des Lernkorpus zu entscheiden, welche Substitutionsmuster in der Grundgesamtheit regelhafter sind als andere, bietet es sich an, die Häufigkeitsrangfolge der Substitutionsmuster in der Grundgesamtheit auch im Lernkorpus einzuhalten. (Vgl. dazu auch die Beschreibung der Prozedur SBSTSTS in Abschnitt 5.4.)

> 4. Forderung: Bildungsmuster, die Linguisten als auf graphischer Ebene regelhaft klassifizieren, soll auch der Lernalgorithmus als regelhaft klassifizieren, falls ihm ein repräsentatives Lernkorpus vorgegeben wird.

Von der Erfüllung der 4. Forderung hängt es ab, inwieweit Linguisten den Algorithmus als Hilfsmittel bei der Beschreibung der Derivationsmorphologie einer Sprache verwenden können.

> 5. Forderung: Der Lernalgorithmus soll zu jedem Lernkorpus eine Menge von Derivationsanweisungen erzeugen, die *redundanzfrei* ist.

Zu jedem Lernkorpus soll also die kürzeste Beschreibung der vorliegenden Ableitungsbeziehung erzeugt werden, die sich in der vorgegebenen formalen Beschreibungssprache formulieren läßt.

Mit der 5. Forderung soll das allgemein in der Linguistik vorherrschende Bestreben nach Beschreibungsökonomie, bzw. Einfachheit der Beschreibung berücksichtigt werden.

3.3 Exkurs: Zum Modellcharakter des Lernalgorithmus sowie zu seinem Bezug zu empiristischen und rationalistischen Spracherwerbstheorien

Die Ausführungen in den Abschnitten 3.1 und 3.2 lassen vielleicht die Frage aufkommen, ob mit dem Lernalgorithmus etwa ein Modell für die Erlernung morphologischer Ableitungsregeln durch den Menschen intendiert ist. Wie bereits in Abschnitt 1.1 skizziert, ist ein solcher Modellcharakter nicht beabsichtigt. Der Anspruch darauf wäre auch nicht zu rechtfertigen. Dies sei hier genauer begründet: Wie Pinker (1979) in einer Überblicksdarstellung über, auch computative Modelle des Spracherwerbs ausführt, müssen solche Modelle zumindest die 6 folgenden *notwendigen*, aber *keineswegs hinreichenden* empirischen Bedingungen erfüllen, um als glaubhafte Modelle bestehen zu können (vgl. Pinker 1979: 218):

1. Das Modell muß der Tatsache Rechnung tragen, daß Sprachen erlernt werden können.
2. Das Modell muß einzelsprachunabhängig sein, d. h., es muß beliebige Sprachen erlernen können.
3. Das Modell muß eine Sprache innerhalb einer Zeitspanne erlernen können, die normalerweise auch von Kindern für den Erwerb der grundlegenden Sprachausschnitte benötigt wird, d. h. innerhalb eines Zeitraumes von 3 Jahren.
4. Das Modell darf als Input nicht solche Arten und/oder Mengen von Informationen verlangen, die dem Kind nicht verfügbar sind.
5. Das Modell sollte solche Vorhersagen über die Zwischenstufen des Spracherwerbs machen, die mit den Ergebnissen von empirischen Untersuchungen der Kindersprache übereinstimmen.
6. Das Modell sollte konsistent sein mit den verfügbaren Erkenntnissen über die kognitiven Fähigkeiten des Kindes.

Wie sich bereits jetzt nach der Ausarbeitung der Problemstellung feststellen läßt, kann der intendierte Lernalgorithmus schon allein deshalb nicht Modell für das Erlernen morphologischer Ableitungsregeln beim Erstspracherwerb sein, weil er zumindest die 2 folgenden der 6 notwendigen Bedingungen nicht erfüllt:

- Bedingung 4 wird nicht erfüllt, da Kindern das für den Erwerb morphologischer Ableitungsregeln erforderliche sprachliche Ausgangsmaterial nicht in solch konzentrierter Form verfügbar ist, wie es dem Lernalgorithmus mit den Wortpaarlisten vorgegeben werden soll. Vielmehr erfahren Kinder die Wörter, die an einer morphologischen Ableitungsbeziehung beteiligt sind, verstreut über Diskurse.
- Bedingung 6 wird schon allein aufgrund der trivialen Tatsache nicht erfüllt, daß Kinder während des Erstspracherwerbs noch nicht lesen können. Der Lernalgorithmus arbeitet jedoch mit Schriftsprache.

Wie die Beschreibung des in dieser Arbeit entwickelten Algorithmus in den Abschnitten 4 und 5 zeigen wird und wie weiterhin verschiedene Anwendungen des Algorithmus ergeben haben, sind auch die Bedingungen 3 und 5 nicht erfüllt:

- Bedingung 3 wird nicht erfüllt, da der Lernalgorithmus morphologische Ableitungsregeln innerhalb einer Zeitspanne erlernt, die mit Sicherheit wesentlich kürzer ist als der von Kindern für dieses Problem benötigte Zeitraum. So benötigt der Algorithmus ca. 5 CPU-Sekunden, um aus 300 Wortpaaren zur Motion[9] im Französischen die entsprechenden Derivationsanweisungen zu erlernen (Diese Wortpaarliste enthält Paare wie

  ('acheteur' <Käufer>, 'acheteuse' <Käuferin>),
  ('acteur' <Schauspieler>, 'actrice' <Schauspielerin>)).

- Schließlich wird Bedingung 5 zumindest deswegen nicht erfüllt, weil der Algorithmus beim Darbieten weiterer Wortpaare eine bereits früher erlernte Menge von Derivationsanweisungen nicht etwa modifiziert, sondern die Menge verwirft und eine völlig neue Menge von Derivationsanweisungen - ausgehend von den bereits früher und den neu dargebotenen Wortpaaren - konstruiert.

9 Unter *Motion* wird in der Linguistik allgemein diejenige Ableitungsbeziehung verstanden, die zwischen den Bezeichnungen männlicher Lebewesen und den entsprechenden Bezeichnungen weiblicher Lebewesen besteht.

Die Bedingungen 1 und 2 werden dagegen - zumindest im Hinblick auf die Derivationsmorphologie - von unserem Algorithmus erfüllt.

Theorien zum Spracherwerb werden in der Linguistik i. allg. in empiristische (auch: behavioristische) und rationalistische (auch: nativistische) Theorien unterschieden. Noam Chomsky charakterisiert in den "Aspects of the theory of syntax" diese beiden Gruppen von Theorien folgendermaßen:

Die *empiristischen Theorien* nehmen an,

> "that only the procedures and mechanisms for the acquisition of knowledge constitute an innate property of the mind." (Chomsky 1965: 51)

Dagegen gehen die *rationalistischen Theorien* davon aus,

> "that the general form of a system of knowledge is fixed in advance as a disposition of the mind, and the function of experience is to cause this general schematic structure to be realized and more fully differentiated." (Chomsky 1965: 51f.)

Fragt man sich, von welcher dieser beiden Positionen in der vorliegenden Arbeit ausgegangen wird, so ergibt sich schon aufgrund der vorangehenden Darlegung der Problemstellung eine klare Orientierung an der rationalistischen Position, denn mit der Definition der 4 Typen von Derivationsanweisungen und mit der Definition der Ordnung von Mengen von Derivationsanweisungen wird bereits die allgemeine Form vorgegeben, in der der Lernalgorithmus die zu erlernenden Kenntnisse darstellen soll. Der zu konstruierende Lernalgorithmus ist also nicht allein in seinen Lernstrategien prädeterminiert, wie es bei einer empiristischen Ausgangsposition der Fall wäre. Vielmehr sind - mit den genannten Definitionen - auch die möglichen internen Repräsentationen der zu erlernenden Daten von vornherein festgelegt.

4 ÜBERBLICK ÜBER DAS PROGRAMMSYSTEM PRISM

Dieser Abschnitt gibt einen Überblick über das Programmsystem, das die in 3.1 wiedergegebene Problemstellung lösen und die in 3.2 gestellten Forderungen erfüllen soll. Eine Detailbeschreibung wird in Abschnitt 5 folgen. Bei der Darstellung in diesem und dem folgenden Abschnitt werden wir uns primär mit der internen Struktur des Programmsystems befassen. Sein Input-Output-Verhalten wird später in Abschnitt 7 sowie im Anhang behandelt werden. Eine Benutzeranleitung soll in dieser Arbeit nicht gegeben werden. Wir verweisen dafür auf den "PRISM User's Guide" (Wothke 1984).

Das Programm trägt den Namen PRISM, der Akronym ist für '*PR*ogram for the *I*nference and *S*imulation of *M*orphological rules'. Es ist in der Programmiersprache PL/1[1] geschrieben und ist z. Z. auf der IBM 3081 (Betriebssystem: MVS) des Regionalen Hochschulrechenzentrums der Universität Bonn implementiert. Das Programm ist für den Stapelbetrieb konzipiert, kann jedoch verhältnismäßig leicht in ein interaktives System überführt werden.

Gemäß dem in Abschnitt 1.1 eingeführten Basismodell besteht PRISM aus 3 Hauptkomponenten:

- Einem *Lernalgorithmus*, der auf der Grundlage eines vom Benutzer vorgegebenen Lernkorpus Derivationsanweisungen erlernt und diese in einer Wissensbasis abspeichert,
- der *Wissensbasis*, in der für beliebig viele morphologische Ableitungsbeziehungen langfristig Mengen von Derivationsanweisungen abgespeichert werden können,
- sowie einem *Anwendungsalgorithmus*, der in der Wissensbasis

1 Die Sprachbeschreibung der verwendeten PL/1-Version findet sich in IBM (1976a).

vorliegende Mengen von Derivationsanweisungen auf vom Benutzer gegebene Quellstrings anwendet und die dadurch erzeugten Zielstrings ausgibt.

Um die Benutzerfreundlichkeit des Programmsystems zu erhöhen, wurde es nicht nur mit den Funktionen versehen, die für die Lösung des gestellten Problems erforderlich schienen, sondern um einige Prozeduren erweitert, die u. a. dem Benutzer eine elegante Steuerung des Programms mittels Kontrolloptionen erlauben und die Eingaben des Benutzers auf ihre syntaktische Wohlgeformtheit und ihre Plausibilität überprüfen. Es ergab sich somit ein Programmsystem, das aus 12 Prozeduren besteht, die entweder dem Lernalgorithmus oder dem Anwendungsalgorithmus oder beiden angehören oder die zur Erhöhung der Benutzerfreundlichkeit dienen. Die Funktion und das Zusammenspiel dieser Prozeduren soll im folgenden knapp skizziert werden. Dabei wollen wir uns als Hilfsmittel der Abbildung 4.1 bedienen:

Hauptprogramm von PRISM ist MONITOR. Bei einer Aktivierung von PRISM ruft MONITOR zunächst die Prozedur GETOPTN auf. GETOPTN liest die vom Benutzer vorgegebenen Steueroptionen zur Steuerung des Systems ein und überprüft diese Benutzereingabe auf syntaktische Wohlgeformtheit und Plausibilität. Falls keine Fehler des Benutzers vorliegen, werden die vom Benutzer angegebenen Optionen - soweit erforderlich - um Standardoptionen ergänzt und alle Optionen werden an den MONITOR übergeben. In Abhängigkeit von den Optionen aktiviert MONITOR eine der Komponenten des Systems und koordiniert die Zusammenarbeit der Prozeduren, aus denen sich die gerade aktivierte Komponente zusammensetzt. 3 Komponenten stehen zur Wahl:

1. Die *Lernkomponente*: Sie setzt sich aus den Prozeduren CHKCRPS, DISCOV, STMTOUT, TODSET, INVERT und LLTMTCH zusammen. Der eigentliche Lernprozess wird dabei von der Prozedur DISCOV ausgeführt, während die anderen Prozeduren eher Hilfsfunktionen haben.
2. Die *Anwendungskomponente*: Ihre Teilprozeduren sind FRODSET, APPLY, DERIVE, LISTOUT und INVERT. Kernstück der Anwendungskomponente ist DERIVE.

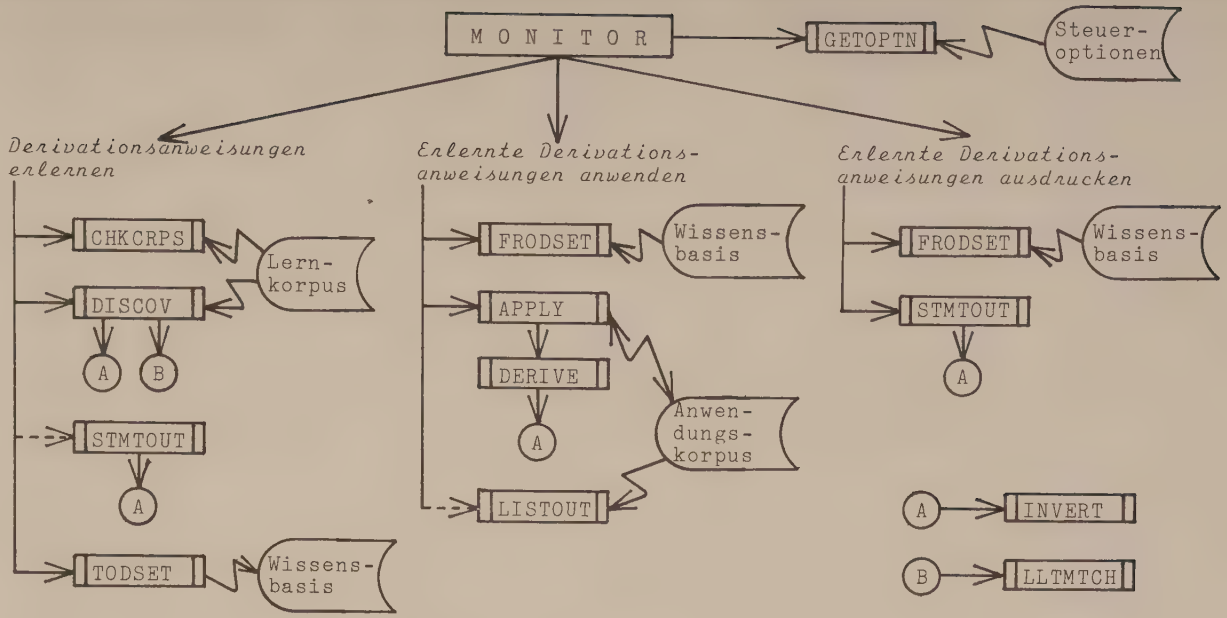


Abb. 4.1: Makrostruktur von PRISM[2].

2 Da diese Graphik lediglich die Makrostruktur des Programms veranschaulichen soll, wird hier von einigen Gegebenheiten des Programms abstrahiert. So werden in der Graphik z. B. die Druckerausgabe und die Arbeitsdateien sowie die Übergabe von Daten (insbesondere von Derivationsanweisungen) im Zentralspeicher zwischen den Teilprogrammen nicht wiedergegeben. Weiterhin treten natürlich Prozeduren und Dateien, die in der Graphik zur Übersichtlichkeit mehrfach angegeben sind (wie z. B. FRODSET und die Wissensbasis), in dem System nur einmal auf.

3. Neben der Lern- und der Anwendungskomponente wurde zur Erhöhung der Benutzerfreundlichkeit noch eine dritte, eher marginale Komponente eingerichtet. Sie druckt lediglich Derivationsanweisungen aus, welche bei einer früheren Aktivierung der Lernkomponente erlernt und in der Wissensbasis abgespeichert wurden. Diese Komponente besteht aus den Prozeduren FRODSET, STMTOUT und INVERT.

Bei *Aktivierung der Lernkomponente* ruft MONITOR zunächst die Prozedur CHKCRPS auf. Sie überprüft ein vom Benutzer vorgegebenes Lernkorpus, das auf einem externen Datenträger vorliegt, auf syntaktische Wohlgeformtheit und Plausibilität. Beim Vorliegen von Fehlern werden entsprechende Fehlermeldungen ausgegeben, und die übrigen Prozeduren der Lernkomponente werden nicht mehr aktiviert. Es werden dann also auch keine Derivationsanweisungen generiert. Weist ein Lernkorpus dagegen keine Fehler auf, so aktiviert MONITOR als nächstes die Prozedur DISCOV. DISCOV führt - ausgehend vom Lernkorpus - die Lernprozesse durch, die zur Erlernung von Derivationsanweisungen erforderlich sind und speichert die erlernten Derivationsanweisungen in einem Strukturarray mit dem Namen STMT ab, auf das jede der anschließend aufgerufenen Prozeduren der Lernkomponente zugreifen kann. Die Form der Derivationsanweisungen in diesem Strukturarray ist nicht identisch mit der Form, die in Abschnitt 3.1 in den Definitionen der 4 verwendeten Typen von Derivationsanweisungen gegeben wurde, sie läßt sich jedoch ohne Informationsverlust in diese Form überführen. Innerhalb von PRISM werden die Derivationsanweisungen in einer Form dargestellt, die ihre schnelle maschinelle Verarbeitung erlaubt. Diese interne Form ist allerdings für den Benutzer schwer lesbar. Daher werden die Derivationsanweisungen in die in den Definitionen angegebene Form transformiert, wenn sie auf den Schirm oder den Drucker ausgegeben werden. Soweit es für das Verständnis unserer weiteren Ausführungen erforderlich ist, werden wir zur besseren Unterscheidung Derivationsanweisungen, wenn sie in der innerhalb von PRISM verwendeten Form vorliegen, *interne Derivationsanweisungen* nennen. Dagegen nennen wir die zur Drucker- oder Schirmausgabe aufbereiteten Anweisungen *externe*

Derivationsanweisungen. Auf die Form der internen Derivationsanweisungen soll in dieser Arbeit nicht eingegangen werden, da es sich nur um ein programmtechnisches Detail handelt, das die im eigentlichen Erkenntnisinteresse liegende Lernstrategie von PRISM kaum beeinflußt. Nachdem DISCOV eine Menge von Derivationsanweisungen erlernt und in dem Strukturarray STMT abgespeichert hat, aktiviert MONITOR die Prozedur STMTOUT, falls eine Steueroption für das Ausdrucken von Derivationsanweisungen vorliegt. STMTOUT greift auf die von DISCOV erzeugten internen Derivationsanweisungen zu und gibt sie - transformiert in ihre externe Repräsentation - auf dem Schirm oder dem Drucker aus. Dabei kann ggf. auch eine Statistik mit ausgegeben werden, die für jede Derivationsanweisung angibt, auf wieviele Quellstrings des Lernkorpus sie anwendbar ist. An dieser Statistik läßt sich bei repräsentativ zusammengestellten Lernkorpora leicht erkennen, welche Derivationsanweisungen Bildungsmuster beschreiben, die aus linguistischer Sicht eher regelhaft sind, und welche eher Ausnahmen beschreiben. Zum Abschluß einer Lernphase ruft MONITOR die Prozedur TODSET auf, die die erlernten internen Derivationsanweisungen langfristig in der Wissensbasis abspeichert. Die Wissensbasis liegt auf einem externen Datenträger vor (z. B. Magnetplatte oder Magnetband). Die in der Wissensbasis gespeicherten Derivationsanweisungen können bei einem späteren Aufruf der Komponente zur Anwendung von Derivationsanweisungen oder der Komponente zum Ausdrucken von Derivationsanweisungen weiterverarbeitet werden.

Die *Aktivierung der Anwendungskomponente* setzt voraus, daß in der Wissensbasis Derivationsanweisungen vorliegen, die zuvor von der Lernkomponente darin abgespeichert wurden. MONITOR aktiviert zunächst FRODSET, das die Derivationsanweisungen aus der Wissensbasis in das Strukturarray STMT einliest, in dem sie dann für die Anwendung auf Quellstrings zur Verfügung stehen. Die anschließend aufgerufene Prozedur APPLY liest vom Benutzer vorgegebene Quellstrings ein, die in einem sogenannten *Anwendungskorpus* auf einem externen Datenträger vorliegen müssen. Jedesmal nachdem APPLY einen Quellstring eingelesen hat, läßt es zu diesem

Quellstring einen Zielstring erzeugen. Dafür ruft APPLY die Funktionsprozedur DERIVE mit dem Quellstring als Argument auf. DERIVE wendet dann die im Strukturarray vorliegenden Derivationsanweisungen auf den Quellstring an und erzeugt dadurch einen Zielstring, den es an APPLY zurückgibt (Falls eine Blockierungsanweisung anwendbar ist, wird allerdings ein leerer Zielstring erzeugt, was der Erzeugung keines Zielstrings gleichkommt). APPLY speichert den erzeugten Zielstring mit dem vorgegebenen Quellstring im Anwendungskorpus ab und geht zur Verarbeitung des nächsten Quellstrings über. Nachdem auf diese Weise schließlich zu allen Quellstrings des Anwendungskorpus Zielstrings erzeugt worden sind, ruft MONITOR abschließend LISTOUT auf, falls eine entsprechende Steueroption zum Ausdrucken des bearbeiteten Anwendungskorpus vorliegt. LISTOUT liest die Quell- und Zielstrings des Anwendungskorpus ein und gibt sie dann - übersichtlich formatiert - auf den Schirm oder den Drucker aus.

Bei *Aktivierung der Komponente zum Ausdrucken von Derivationsanweisungen*, die bei einer früheren Programmausführung erlernt wurden, wird zunächst FRODSET aufgerufen, das die Derivationsanweisungen aus der Wissensbasis in das Strukturarray STMT einliest. Anschließend aktiviert MONITOR die Prozedur STMTOUT, die die Anweisungen in der oben bereits skizzierten Weise ausgibt.

Auf 2 Prozeduren wurde in diesem Überblick noch nicht eingegangen: INVERT und LLTMTCH. Es handelt sich hierbei um Funktionsprozeduren, die recht einfache Operationen mit Zeichenketten durchführen. Da die betreffenden Operationen in PRISM häufiger und an verschiedenen Programmstellen benötigt werden, bot es sich an, sie in Form separater Prozeduren zu realisieren, die dann an den betreffenden Programmstellen aktiviert werden. Die Prozedur INVERT invertiert eine Zeichenkette oder einen linken Teilstring einer Zeichenkette, die ihr als Argument übergeben wird, und gibt die Inversion als Resultat an die aufrufende Prozedur zurück. An LLTMTCH werden von der jeweils aufrufenden Prozedur 2 Zeichenketten als Argumente übergeben. LLTMTCH bestimmt zu den 2 Zeichenketten die Länge der längsten linken Übereinstimmung und meldet diese Länge der aufrufenden Prozedur.

5 DIE PROZEDUREN DES PROGRAMMSYSTEMS

In diesem Abschnitt werden die einzelnen Prozeduren des Programmsystems PRISM detaillierter beschrieben. Unser Hauptinteresse gilt dabei den Prozessen, die die maschinelle Erlernung von Derivationsanweisungen aus einem Lernkorpus erlauben. Den größten Raum wird daher die Darstellung der Prozedur DISCOV einnehmen (Abschnitt 5.4). Die übrigen Prozeduren sind zwar für das Funktionieren des Gesamtsystems erforderlich, bergen jedoch kaum wesentlich neue Erkenntnisse in sich. Trotzdem kann auf eine Kurzbeschreibung dieser Prozeduren nicht verzichtet werden: Zum einen setzt das Verständnis von DISCOV nämlich die Kenntnis einiger dieser Prozeduren voraus. Zum anderen liefert die Dokumentation des gesamten Systems notwendige Informationen für einen Nachbau des Systems sowie für die Entwicklung eines verbesserten Systems auf der Grundlage von PRISM.

Sicherlich sind nicht alle Ausführungen in diesem Abschnitt für alle Leser gleichermaßen wichtig:

- Wer primär an den Lernstrategien des Systems interessiert ist, für den dürfte in erster Linie die Beschreibung von DISCOV (5.4) und ggf. von STMTOUT (5.5) und TODSET (5.6) relevant sein. Die anderen Abschnitte kann er durchaus überspringen.
- Für das Verständnis des Anwendungsalgorithmus genügt die Lektüre der Beschreibungen von FRODSET (5.7), APPLY (5.8) und insbesondere von DERIVE (5.9).
- Wer das System oder Ausschnitte davon dagegen nachbauen oder erweitern möchte, sollte sich möglichst mit den Beschreibungen aller Prozeduren vertraut machen und unbedingt auch die kommentierte Auflistung des Quellprogramms im Anhang (Abschnitt 10.1) heranziehen.

Abschließend sei noch betont, daß wir hier keine Prozedur bis

ins letzte Detail beschreiben. Eine Detailbeschreibung würde zum einen den Umfang dieses Abschnitts zu sehr aufblähen, zum anderen würde die Übersichtlichkeit der Darstellung weitgehend verloren gehen, und das Wesentliche würde nicht klar genug zum Ausdruck kommen. Soweit sich Detailfragen ergeben, die Abschnitt 5 nicht beantwortet, sei auf die Wiedergabe des Quellprogramms im Anhang verwiesen. Das Quellprogramm dürfte die meisten Detailfragen klären.

5.1 MONITOR

Aufrufe

Aufgerufene Prozeduren: GETOPTN, CHKCRPS, DISCOV, STMTOUT, TODSET, FRODSET, APPLY, LISTOUT.

Eingabedaten

- Steueroptionen von der Prozedur GETOPTN.
- Fehlerkodes von den Prozeduren GETOPTN, CHKCRPS, DISCOV und FRODSET.

Ausgabedaten

- Vorspann zur Programmidentifikation sowie Programmabschlußmeldung auf den Schirm oder den Drucker.
- Steueroption an die Prozedur STMTOUT.
- Ggf. Fehlermeldungen auf den Schirm oder den Drucker.

Funktion und Arbeitsweise

MONITOR ist das Hauptprogramm von PRISM. Es dient im wesentlichen zur Steuerung des Gesamtsystems.

MONITOR gibt zunächst einen Vorspann zur Programmidentifikation auf den Schirm oder den Drucker aus. Anschließend aktiviert es die Prozedur GETOPTN (vgl. Abschnitt 5.2), die die Steueroptionen des Benutzers einliest und an MONITOR übergibt. Falls die Steueroptionen fehlerhaft spezifiziert wurden, gibt GETOPTN einen entsprechenden Fehlerkode zurück, der MONITOR zur Beendigung des Programmablaufs veranlaßt. Andernfalls aktiviert MONITOR die durch die Steueroptionen angesprochene Programmkomponente und koordiniert das Zusammenspiel der Prozeduren, die diese Kom-

ponente konstituieren. Im wesentlichen wurde diese Steuerung bereits beim Überblick über PRISM in Abschnitt 4 beschrieben. Wir wollen daher nur noch auf einige Details hinweisen, die in Abschnitt 4 vernachlässigt wurden.

- Die Prozeduren CHKCRPS, DISCOV und FRODSET geben an MONITOR einen Fehlerkode zurück, falls fehlerhafte Eingaben des Benutzers vorliegen oder falls beim Aufruf des System-Hilfsprogramms SyncSort[1], das zum Sortieren von Daten dient, ein Fehler auftritt. Der Fehlerkode veranlaßt MONITOR zur Beendigung des Programmablaufs.
- Die Prozedur STMTOUT, die zur Transformation von internen in externe Derivationsanweisungen dient und die externen Derivationsanweisungen auf dem Schirm oder dem Drucker ausgibt, wird mit einer Steueroption als Argument aufgerufen. Diese von GETOPTN an MONITOR übergebene Option, informiert STMTOUT darüber, ob die Derivationsanweisungen mit oder ohne statistische Angaben ausgegeben werden sollen.

Sobald die von MONITOR aktivierte Komponente ihre Arbeit beendet hat, wird die Kontrolle an MONITOR zurückgegeben, das dann eine Programmabschlußmeldung auf den Schirm oder den Drucker schickt.

Während des gesamten Programmablaufs kontrolliert MONITOR für alle verwendeten Files (außer SYSPRINT), ob in den aufgerufenen Prozeduren bei der Arbeit mit diesen Files ein Fehler auftritt. Falls ja, produziert MONITOR eine Fehlermeldung und beendet den Programmablauf.

5.2 GETOPTN

Aufrufe

Aufrufende Prozedur: MONITOR.

Eingabedaten

Vom Benutzer vorgegebene Steueroptionen von externem Datenträger.

1 Vgl. IBM 1976b: 221ff. u. Whitlow 1980.

Ausgabedaten

- Steueroptionen und ggf. Fehlerkode an die Prozedur MONITOR.
- Informations- und ggf. Fehlermeldungen zu den Steueroptionen auf den Schirm oder den Drucker.

Funktion und Arbeitsweise

GETOPTN wird vom Hauptprogramm MONITOR aktiviert. Es stellt die Optionen zur Steuerung von PRISM bereit und übergibt sie an MONITOR, das dann in Abhängigkeit von diesen Optionen den weiteren Ablauf von PRISM steuert.

Zunächst liest GETOPTN von einer externen Datei das Datenfeld ein, das die vom Benutzer vorgegebenen Steueroptionen enthält.

Es gibt 2 Arten von Steueroptionen:

- Mit Hilfe von 3 Optionen kann gesteuert werden, welche der 3 Komponenten des Systems aktiviert werden soll.
- Zusätzlich können Optionen angegeben werden, welche Art und Umfang der Druckerausgabe der aktivierten Komponente spezifizieren.

Nicht alle Optionen muß der Benutzer unbedingt explizit angeben, da beim Weglassen von Optionen Standardoptionen in Kraft treten. Die Optionen und die Syntax zu ihrer Kombination sollen hier nicht im einzelnen erläutert werden. Wir verweisen dafür auf den "PRISM User's Guide" (Wothke 1984: 9-11) und die Beschreibung der Optionen zu Anfang von Abschnitt 10.1.

Nachdem GETOPTN das Datenfeld eingelesen hat, überprüft es dieses auf formale Fehler und gibt dabei auf den Schirm oder den Drucker verschiedene Informationsmeldungen aus sowie zu jedem Fehler eine Fehlermeldung. Abschließend werden die explizit angegebenen Optionen - soweit erforderlich - um Standardoptionen ergänzt, und alle Optionen werden an MONITOR übergeben. Weist das Datenfeld allerdings Fehler auf, so wird anstatt der Optionen ein Fehlerkode an MONITOR übergeben, der die Beendigung des Programmablaufs auslöst.

5.3 CHKCRPS

Aufrufe

Aufrufende Prozedur: MONITOR.

Eingabedaten[2]

Vom Benutzer vorgegebenes Lernkorpus von externem Datenträger.

Ausgabedaten[2]

- Ggf. Fehlerkode an die Prozedur MONITOR.
- Informations- oder Fehlermeldungen zum Lernkorpus auf den Schirm oder den Drucker.

Funktion und Arbeitsweise

Die Prozedur CHKCRPS wird vom Hauptprogramm MONITOR als erste Prozedur im Rahmen der Lernkomponente aktiviert. Sie liest ein vom Benutzer vorgegebenes Lernkorpus aus einer externen Datei ein und überprüft es auf formale Fehler. Dabei wird das Korpus - soweit möglich - auf Übereinstimmung mit den in Abschnitt 3.1 eingeführten Regeln (Quellstring darf nicht leer sein, Rechtseindeutigkeit) sowie mit weiteren formalen Konventionen, die hier nicht genannt werden sollen,[3] überprüft. Beim Vorliegen von Fehlern wird auf den Schirm oder den Drucker für jeden Fehler eine entsprechende Fehlermeldung ausgegeben, und es wird an MONITOR mit einem Fehlerkode das Vorliegen von Fehlern gemeldet. Der Fehlerkode bewirkt, daß MONITOR unmittelbar nach der Ausführung von CHKCRPS den Programmablauf beendet. Die übrigen Prozeduren der Lernkomponente (DISCOV, STMTOUT und TODISK) werden dann also nicht mehr aktiviert, und es werden auch keine Derivationsanweisungen zu dem fehlerhaften Lernkorpus erlernt. Ist das Lernkorpus fehlerfrei, so gibt CHKCRPS eine entsprechende Informationsmeldung aus und gibt - ohne Übergabe eines Fehlerkodes - die Kontrolle an MONITOR zurück, das dann die Aktivierung der Lernkomponente fortsetzt.

2 Die Ein- und Ausgabe für eine Arbeitsdatei, die nur innerhalb von CHKCRPS benötigt wird, wird hier nicht erwähnt, da sie für das wesentliche Verständnis von CHKCRPS nicht erforderlich ist.

3 Die formalen Konventionen für den Aufbau von Lernkorpora sind vollständig im "PRISM User's Guide" (Wothke 1984: 12) genannt.

5.4 DISCOV

Aufrufe

Aufrufende Prozedur: MONITOR.

Aufgerufene Prozeduren: INVERT, LLTMTCH.

Eingabedaten[4]

Vom Benutzer vorgegebenes Lernkorpus von externem Datenträger.

Ausgabedaten[4]

- Interne Derivationsanweisungen mit Häufigkeitsangaben in das Strukturarray STMT.
- Ggf. eine Fehlermeldung auf den Schirm oder den Drucker.
- Ggf. Fehlerkode an die Prozedur MONITOR.

Funktion und Arbeitsweise

DISCOV erlernt zu einem vom Benutzer vorgegebenen Lernkorpus Derivationsanweisungen. Es ist somit wichtigster Bestandteil der Lernkomponente vom PRISM, und es ist daher auch diejenige Prozedur, deren Entwicklung und Evaluierung Hauptziel dieser Arbeit war. Funktion und Arbeitsweise von DISCOV sollen daher hier sehr ausführlich beschrieben werden.

DISCOV wird von MONITOR im Rahmen der Lernkomponente aktiviert, nachdem CHKCRPS das Lernkorpus auf formale Fehler überprüft hat. Die Aktivierung von DISCOV erfolgt allerdings nur, falls CHKCRPS feststellt, daß das Lernkorpus fehlerfrei ist.

DISCOV setzt sich im wesentlichen aus 4 *internen* Prozeduren zusammen (vgl. Abb. 5.1):

1. Zunächst wird die Prozedur PREPCRP durchlaufen. Sie bereitet ein Lernkorpus für die Erlernung von Derivationsanweisungen auf.
2. Anschließend konstruiert die Prozedur BLCKSTS ausgehend vom Lernkorpus Blockierungsanweisungen und speichert sie in dem Strukturarray STMT ab. Dieses Array wird während der gesam-

4 Die Ein- und Ausgabe für 3 Arbeitsdateien wird hier nicht erwähnt, da diese Dateien nur innerhalb von DISCOV verwendet werden und DISCOV über sie keine Daten von anderen Prozeduren empfängt oder an diese weitergibt.

ten Ausführung von PRISM im Zentralspeicher gehalten. Es dient zur Übergabe von Derivationsanweisungen zwischen denjenigen Prozeduren von PRISM, die auf Derivationsanweisungen zugreifen.

3. Die Substitutionsanweisungen werden dann von der Prozedur SBSTSTS konstruiert und ebenfalls in STMT - hinter den Blokkierungsanweisungen - abgespeichert.
4. Schließlich bereitet die Prozedur PREPSTS die erlernten Derivationsanweisungen für die Weiterverarbeitung durch STMTOUT und TODSET sowie durch die beiden anderen Hauptkomponenten von PRISM auf.

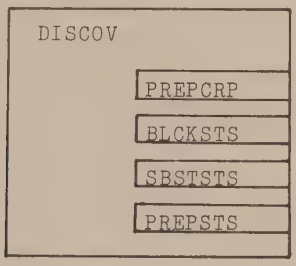


Abb. 5.1.: Nassi-Shneiderman-Diagramm zur Makrostruktur von DISCOV.

Die Beschreibung von DISCOV wollen wir nach den 4 internen Prozeduren gliedern.

PREPCRP

Diese Prozedur baut ausgehend vom Lernkorpus 2 Arbeitsdateien auf, die jeweils alleinige Grundlage für die Erlernung der Blokkierungsanweisungen in BLCKSTS und der Substitutionsanweisungen in SBSTSTS sind. Auf das ursprüngliche Lernkorpus wird dann in BLCKSTS und in SBSTSTS nicht mehr zugegriffen. Zur leichteren Identifizierung nennen wir die 2 Arbeitsdateien hier WORK1 und WORK2.

WORK1 bildet die Grundlage für die Erlernung von Blockierungsanweisungen durch die Prozedur BLCKSTS. Die Datei wird ausgehend von denjenigen Wortpaaren des Lernkorpus erstellt, die

keinen Zielstring haben. Nur von diesen Wortpaaren ausgehend lassen sich nämlich Blockierungsanweisungen erlernen. Die Wortpaare mit nicht-leerem Zielstring erfordern dagegen die Erlernung von Substitutionsanweisungen. Zu jedem Quellstring mit leerem Zielstring bildet PREPCRP einen sogenannten Analyserecord, der in WORK1 abgespeichert wird. Dieser Analyserecord enthält zu dem Quellstring in festen Feldern u. a. das Impediens einer möglicherweise von BLCKSTS zu erlernenden präfixorientierten Blockierungsanweisung und das Impediens einer möglicherweise zu erlernenden suffixorientierten Blockierungsanweisung. Wir verwenden hier das Adverb 'möglicherweise', weil zu jedem Quellstring, der keinen Zielstring aufweist, *entweder* nur eine präfix- *oder* nur eine suffixorientierte Blokkierungsanweisung erlernt wird. Welche von beiden schließlich zum Zuge kommt, entscheidet BLCKSTS (s. u.) aufgrund der in WORK1 vorliegenden Daten. Weiterhin enthält der Analyserecord zu jedem der 2 Impedienten die Angabe, wie häufig es in den Analyserecords von WORK1 auftritt. Als Impediens der potentiellen präfixorientierten Blockierungsanweisung ermittelt PREPCRP den kürzesten *linken* Teilstring, in dem der vorliegende Quellstring *ohne* Zielstring mit keinem der Quellstrings *mit* Zielstring übereinstimmt. Entsprechend wird als Impediens der potentiellen suffixorientierten Blockierungsanweisung der kürzeste *rechte* Teilstring ermittelt, in dem der vorliegende Quellstring *ohne* Zielstring mit keinem der Quellstrings *mit* Zielstring übereinstimmt. Zur Auffindung der potentiellen Impedienten ruft DISCOV die Hilfsprozedur LLTMTCH auf (vgl. Abschnitt 5.12). Enthält ein Lernkorpus neben Quellstrings *ohne* Zielstring keine Quellstrings *mit* Zielstring, so werden als potentielle Impedienten Nullstrings gewählt. Mit der Ermittlung der *kürzesten* linken und rechten Teilstrings als potentielle Impedienten leistet PREPCRP einen notwendigen Beitrag zur Erfüllung der Forderung nach *Redundanzfreiheit* von Mengen von Derivationsanweisungen (vgl. Forderung 5 in Abschnitt 3.2). Außerdem reichen diese Teilstrings gerade aus, um *adäquat* zu beschreiben, daß zu dem vorliegenden Quellstring *ohne* Zielstring kein Zielstring erzeugt werden soll. Da sie zu keinem der Quellstrings *mit* Zielstring linker, bzw.

rechter Teilstring sind, ist weiterhin die Blockierungsanweisung, die später BLCKSTS mit einem dieser Teilstrings als Impediens bildet, auf keinen der Quellstrings *mit* Zielstring anwendbar, was ja auch erforderlich ist, da sonst fälschlicherweise zu einem solchen Quellstring kein Zielstring erzeugt werden könnte.

Da die Impedienten stets die kürzesten linken, bzw. rechten Teilstrings sind, in denen die Quellstrings *ohne* Zielstring mit keinem der Quellstrings *mit* Zielstring übereinstimmen, können sie niemals echte linke (bei präfixorientierten Impedienten) oder rechte (bei suffixorientierten Impedienten) Teilstrings voneinander sein. Dies läßt sich leicht beweisen. Wir führen den Beweis indirekt, und zwar nur für die potentiellen präfixorientierten Impedienten. Für die potentiellen suffixorientierten Impedienten verläuft er analog.

Nehmen wir an, X ist der kürzeste linke Teilstring, in dem ein Quellstring Q_1 *ohne* Zielstring mit keinem der Quellstrings *mit* Zielstring übereinstimmt.

- 1. Fall: Es existiere ein weiteres präfixorientiertes Impediens Y, zu dem X echter linker Teilstring ist.

  Dies ist nicht möglich, da der Quellstring Q_2 *ohne* Zielstring, dessen kürzeste Nichtübereinstimmung mit den Quellstrings *mit* Zielstring Y ist, dann auch auf X beginnen müßte. Wenn aber der Teilstring X bereits kürzeste linke Nichtübereinstimmung für Q_1 war, dann muß er es auch für Q_2 sein. Y wäre also nicht die kürzeste linke Nichtübereinstimmung.

- 2. Fall: Es existiere ein weiteres präfixorientiertes Impediens Y, das echter linker Teilstring von X ist.

  Auch dies ist nicht möglich, da dann *nicht* X der kürzeste linke Teilstring wäre, in dem Q_1 mit keinem der Quellstrings *mit* Zielstring übereinstimmt, *sondern* Y.

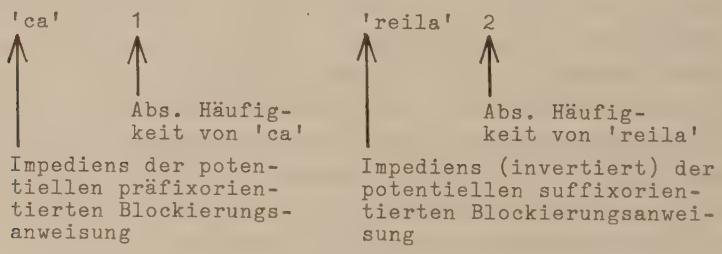
Da also 2 präfixorientierte, bzw. 2 suffixorientierte Impedienten niemals Teilstrings voneinander sein können, ist auch - wie bereits in 3.1 erwähnt - die relative Anwendungsreihenfolge von Blockierungsanweisungen untereinander beliebig.

Das Verfahren, mit dem PREPCRP zu einem Quellstring *ohne*

Zielstring einen Analyserecord konstruiert, sei nun an einem Beispiel veranschaulicht. Nehmen wir an, die folgende (linguistisch nicht repräsentative) Wortpaarliste zur Motion im Französischen wird als Lernkorpus vorgegeben:

```
('chapelier' <Hutmacher>,       'chapelière' <Hutmacherin>),
('infirmier' <Krankenpfleger>, 'infirmière' <Krankenschwester>),
('cavalier'  <Reiter>,          ''),
('chevalier' <Ritter>,          '').
```

Die Quellstrings 'cavalier' und 'chevalier' haben leere Zielstrings, da zu ihnen keine weibliche Form existiert. PREPCRP muß daher zu diesen beiden Quellstrings jeweils einen Analyserecord für die Erlernung von Blockierungsanweisungen erstellen und in WORK1 ablegen. Wir beschränken uns hier darauf, die Erstellung des Analyserecords zu 'cavalier' zu beschreiben. Als Impediens für die potentielle präfixorientierte Blockierungsanweisung wird 'ca' ermittelt, da dies der kürzeste linke Teilstring ist, in dem 'cavalier' mit keinem der vorliegenden Quellstrings mit nicht-leerem Zielstring, also mit 'chapelier' und 'infirmier', übereinstimmt. Impediens der potentiellen suffixorientierten Blockierungsanweisung wird 'alier', das der kürzeste rechte Teilstring ist, in dem 'cavalier' weder mit 'chapelier' noch mit 'infirmier' übereinstimmt. 'alier' würde auch im Analyserecord zu 'chevalier' als Impediens der möglichen suffixorientierten Blockierungsanweisung ermittelt. Die Angabe zu seiner Häufigkeit in den Analyserecords von WORK1 wird daher auf 2 gesetzt. Häufigkeit des Impediens 'ca' ist 1. Der Analyserecord zu 'cavalier' hat schließlich folgenden Inhalt:



Zu 'chevalier' ergibt sich als zweiter Analyserecord in WORK1:

'che' 1 'reila' 2.

Unklar ist vielleicht, warum das potentielle suffixorientierte Impediens *invertiert* im Analyserecord auftritt. Die gleiche Frage wird sich auch bei der Beschreibung der Datei WORK2 stellen, in der u. a. auch invertierte Quellstrings auftreten: Bei den meisten Operationen, die Quellstrings von rechts her bearbeiten, arbeitet PRISM mit invertierten Zeichenketten, d. h., es werden die Inversionen der Quellstrings verwendet, und auch die Impedienten, Substituenda, Substituenten und Kontextstrings werden invertiert verarbeitet. Die Inversionen werden dabei von der Prozedur INVERT erstellt (vgl. Abschnitt 5.11). Anstatt den Quellstring von rechts nach links zu bearbeiten, wird dann der invertierte Quellstring von links nach rechts bearbeitet. Diese Arbeit mit Inversionen ist logisch nicht zwingend. Es wäre aus Rechenzeitgründen wahrscheinlich sinnvoller gewesen, mit nicht invertierten Zeichenketten zu arbeiten. Wir haben uns bei der Entwicklung von PRISM dennoch für die Verwendung von Inversionen entschieden, da Algorithmen, die von links nach rechts über Zeichenketten operieren, i. allg. leichter zu durchschauen sind als solche, die Zeichenketten von rechts nach links bearbeiten. Dies dürfte wohl damit zusammenhängen, daß wir aufgrund unserer Verwendung der lateinischen Schrift gewohnt sind, von links nach rechts zu schreiben und zu lesen.

Auch zu den Wortpaaren mit nicht-leerem Zielstring stellt PREPCRP Analyserecords auf, die allerdings einen anderen Aufbau und Inhalt haben als die bisher beschriebenen Analyserecords in WORK1. Die Analyserecords zu den Paaren mit nicht-leerem Zielstring werden in der Datei WORK2 abgelegt. Sie bilden die Grundlage für die Erlernung der Substitutionsanweisungen durch die Prozedur SBSTSTS. Jeder Analyserecord in WORK2 enthält zu dem Wortpaar, zu dem er angelegt wird, u. a. den Quellstring sowie die Substituenda und die Substituenten der - ausgehend von dem Wortpaar - zu konstruierenden präfix- und suffixorientierten Substitutionsanweisung. Wie PREPCRP ausgehend von Quell- und Zielstring die Substituenda und Substituenten bestimmt und wel-

chen Inhalt der Analyserecord hat, sei an einem Beispiel erläutert. Gegeben sei das Wortpaar

('produzieren', 'Produktion').

PREPCRP ermittelt zunächst Anfangsposition und Länge der längsten gemeinsamen Zeichenkette von Quell- und Zielstring. Die längste gemeinsame Zeichenkette ist 'rodu'. Anfangsposition in Quell- und Zielstring ist 2, Länge ist 4. Ausgehend von diesen Werten bestimmt PREPCRP Substituenda und Substituenten der präfix- und der suffixorientierten Substitutionsanweisungen, die die Erzeugung von 'Produktion' aus 'produzieren' beschreiben. Als Substituendum und Substituens der zu konstruierenden präfixorientierten Substitutionsanweisung wird jeweils die Zeichenkette identifiziert, die im Quell-, bzw. Zielstring *links* von der längsten gemeinsamen Zeichenkette steht. Substituendum der präfixorientierten Substitutionsanweisung ist also 'p'. Substituens ist 'P'. Dieses Substitutionsmuster beschreibt lediglich die Änderung der Groß/Kleinschreibung für den Anfangsbuchstaben. Entsprechend bilden die 2 Zeichenketten, die in Quell- und Zielstring *rechts* von der längsten gemeinsamen Zeichenkette stehen, das Substituendum und das Substituens der zu konstruierenden suffixorientierten Substitutionsanweisung. Substituendum wird demnach 'zieren', Substituens wird 'ktion'. PREPCRP ermittelt also die *kürzestmöglichen* Substituenda und Substituenten, die in der vorgegebenen Beschreibungssprache zur adäquaten Erzeugung der Ziel- aus den Quellstrings möglich sind. Es stellt somit diejenigen Substituenda und Substituenten auf, die für die Erreichung redundanzfreier Mengen von Derivationsanweisungen notwendig sind (vgl. Forderung 5 in Abschnitt 3.2). Außerdem wird mit diesen kürzesten Substituenda und Substituenten die weitestgehende Generalisierung geleistet, die ohne Verlust der schwachen Adäquatheit (vgl. Forderung 1 in Abschnitt 3.2) möglich ist. Längere Substituenda und Substituenten würden Zeichenketten ersetzen, ohne daß etwas verändert wird, sie wären also redundant. Außerdem wären sie möglicherweise zu speziell, d. h. auf zu wenige Quellstrings anwendbar, so daß u. U. keine starke Adäquatheit (vgl. Forderung 3 in Abschnitt 3.2) erzielt

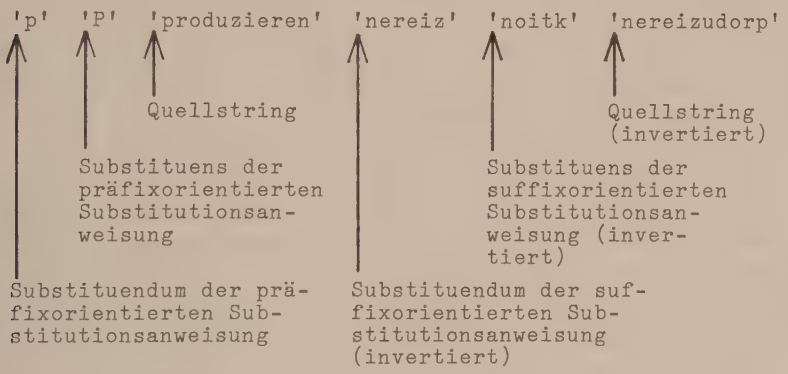
werden kann. Im vorliegenden Beispiel wären etwa das Substituendum 'uzieren' und das Substituens 'uktion'

- *redundant*, da die Ersetzung von 'u' durch 'u' nichts verändert.
- *zu speziell*, da sie - i. U. zu 'zieren' und 'ktion' - beispielsweise für 'vivisezieren' <==> 'Vivisektion' nicht mehr gelten.

Der Analyserecord zum Wortpaar

('produzieren', 'Produktion')

enthält schließlich folgende Elemente:



Der Quellstring wird im Analyserecord mit abgespeichert, da er noch für die spätere Aufstellung der Kontextbedingungen der zu konstruierenden Substitutionsanweisungen erforderlich ist.

BLCKSTS

Nachdem PREPCRP die Arbeitsdateien WORK1 und WORK2 erstellt hat, ermittelt BLCKSTS auf der Grundlage von WORK1 die Blockierungsanweisungen und speichert sie in STMT ab. Zu jedem Quellstring muß nur eine - entweder präfix- oder suffixorientierte - Blokkierungsanweisung konstruiert werden. BLCKSTS muß daher eines der beiden potentiellen Impedienten jedes Analyserecords für die Aufstellung einer Blockierungsanweisung auswählen und dabei darauf achten, daß die entstehende Menge von Blockierungsanweisungen möglichst redundanzfrei wird (vgl. die 5. Forderung in Ab-

schnitt 3.2). Entstehen aufgrund identischer Impedienten in verschiedenen Analyserecords mehrere identische Blockierungsanweisungen, so wird natürlich nur eine von ihnen in die (endgültige) Menge von Blockierungsanweisungen in STMT aufgenommen. Zu dem bei der Beschreibung von PREPCRP als Beispiel angegebenen französischen Lernkorpus lassen sich aufgrund von WORK1 4 verschiedene schwach adäquate Mengen von Blockierungsanweisungen konstruieren:

1. # 'ca'¬
   # 'che'¬
2. # 'ca'¬
   'alier'#¬
3. # 'che'¬
   'alier'#¬
4. 'alier'#¬

Redundanzfrei ist allein die vierte Menge, die nur eine Blokkierungsanweisung enthält. Kürzer läßt sich (in der vorgegebenen Beschreibungssprache) nicht angeben, daß zu 'cavalier' und 'chevalier' kein Zielstring erzeugt werden soll. Auch läßt sich diese Regel nicht weiter generalisieren, ohne daß die schwache Adäquatheit verlorengeht. BLCKSTS soll also auf der Grundlage von WORK1 die vierte Menge von Blockierungsanweisungen konstruieren. Um dies zu erreichen und auch um generell die kürzeste und weitestgehend generalisierende Menge zu bilden, geht BLCKSTS folgendermaßen vor: Mit einem potentiellen präfixorientierten Impediens wird nur dann tatsächlich eine präfixorientierte Blockierungsanweisung gebildet,

- wenn seine Häufigkeit größer ist als die von mindestens einem der potentiellen suffixorientierten Impedienten, mit denen es in den Analyserecords gemeinsam vorkommt,
- oder wenn seine Häufigkeit gleich der Häufigkeit des seltensten potentiellen suffixorientierten Impediens ist, mit dem es gemeinsam in einem Analyserecord steht, *und es gleichzeiti* nicht länger ist als dieses seltenste Impediens.

In allen anderen Fällen wird mit dem potentiellen suffixorien-

tierten Impediens eine suffixorientierte Blockierungsanweisung gebildet.

Mit dieser Methodik bildet BLCKSTS ausgehend von den Daten in WORK1 i. allg. die kürzestmögliche Menge von Blockierungsanweisungen. Angewandt auf die obigen Analyserecords zu 'cavalier' und 'chevalier' würde die kürzestmögliche und allgemeinste Menge von Blockierungsanweisungen gebildet, die lediglich die Anweisung 'alier'#¬ enthält.

Die erlernten Blockierungsanweisungen speichert BLCKSTS im Strukturarray STMT ab. Zu jeder Blockierungsanweisung wird dabei in diesem Strukturarray mit angegeben, auf wieviele Quellstrings des Lernkorpus sie anwendbar ist. Diese Häufigkeitsangabe wird später von der Prozedur STMTOUT beim Ausdrucken von Derivationsanweisungen u. U. mit ausgegeben. Bei repräsentativen Lernkorpora kann der Benutzer an dieser Angabe leicht erkennen, welche Blockierungsanweisungen eher Ausnahmen beschreiben und welche eher linguistische Regeln darstellen.

SBSTSTS

Nachdem BLCKSTS Blockierungsanweisungen erlernt und im Strukturarray STMT abgespeichert hat, wird SBSTSTS aktiviert. SBSTSTS konstruiert ausgehend von den Daten in WORK2 die Substitutionsanweisungen und legt sie in STMT hinter den Blockierungsanweisungen ab.

WORK2 enthält bereits die Substituenda und die Substituenten der zu bildenden Substitutionsanweisungen. Was noch fehlt, sind die Einführung der Kontextbedingungen, die Festlegung der Anwendungsreihenfolge der Substitutionsanweisungen und die Reduzierung mehrerer identischer Substitutionsanweisungen auf eine. SBSTSTS löst diese Aufgabe in 2 Hauptschritten: Zunächst erlernt es ausgehend von WORK2 die präfixorientierten Substitutionsanweisungen. Anschließend werden die suffixorientierten Substitutionsanweisungen erlernt, und zwar nach den *gleichen* Strategien wie die präfixorientierten Anweisungen. Es ist daher hier nicht erforderlich, die einzelnen Lernstrategien des Algorithmus sowohl für die präfix- als auch für die suffixorientierten Substitutionsanweisungen zu erläutern. Vielmehr genügt es, die ver-

schiedenen Teilstrategien jeweils nur für präfix- oder nur für suffixorientierte Anweisungen zu beschreiben.

Das Problem der *Einführung von Kontextbedingungen* sei hier nur für präfixorientierte Substitutionsanweisungen erläutert. Kontextbedingungen müssen nur dann eingeführt werden,

- wenn in den Analyserecords von WORK2 *dasselbe* Substituendum X_1 mit *verschiedenen* Substituenten Y_1, ..., Y_{n-1} auftritt
- sowie wenn in den Analyserecords ein weiteres Substituendum X_2 auftritt, zu dem X_1 *echter linker Teilstring* ist.

In diesem Fall können nicht etwa gleichzeitig die kontext*freien* Substitutionsanweisungen

$$X_1 \rightarrow Y_1/\#_$$
$$X_1 \rightarrow Y_2/\#_$$
$$\vdots$$
$$X_1 \rightarrow Y_{n-1}/\#_$$
$$X_2 \rightarrow Y_n/\#_$$

in die zu erlernende Menge von Derivationsanweisungen aufgenommen werden, da wegen der in Abschnitt 3.1 eingeführten Ordnung von Mengen von Derivationsanweisungen stets nur die erste Anweisung zur Anwendung käme (sie ist nämlich in allen Situationen anwendbar, in denen auch die übrigen Anweisungen anwendbar sind) Damit jede Anweisung potentiell zur Anwendung kommen kann, müssen daher einige Anweisungen um Kontextbedingungen erweitert werden, und zwar um *verschiedene* Kontextbedingungen. Dabei soll darauf geachtet werden, daß die regelhafteren, also die häufigeren Substitutionsmuster der Sprache mit allgemeineren, kontextfreieren Anweisungen beschrieben werden, die einen breiteren Anwendungsbereich haben. Dagegen sollen die selteneren Substitutionsmuster, die eher für sprachliche Ausnahmen gelten, mit spezielleren, kontextsensitiveren Anweisungen beschrieben werden, die durch den höheren Grad an Kontextsensitivität auch einen engeren Anwendungsbereich haben.

Problem ist nun: Wie soll SBSTSTS *auf der Grundlage der Daten in WORK2* entscheiden, welche Substitutionsmuster *in der Sprache*

regelhafter sind und welche eher die Ausnahme sind? Per definitionem ist in *repräsentativen* Lernkorpora die Häufigkeitsrangfolge der Substitutionsmuster identisch mit ihrer Häufigkeitsrangfolge in der betreffenden Sprache (vgl. dazu die Def. *repräsentatives Lernkorpus* in Abschnitt 3.2). Bei repräsentativen Lernkorpora lassen sich daher aufgrund ihrer Häufigkeitsrangfolge regelhaftere Substitutionsmuster leicht von den weniger regelhaften unterscheiden. Anders sieht es bei Lernkorpora aus, die (aus linguistischer Sicht) *zufällig* zusammengestellt sind. Hier kann nicht davon ausgegangen werden, daß sich aus der Häufigkeitsrangfolge der Substitutionsmuster *immer* der Grad ihrer Regelhaftigkeit in der betrachteten Sprache ablesen läßt. Es scheint allerdings plausibel anzunehmen, daß in zufällig zusammengestellten Lernkorpora die Häufigkeitsrangfolge der Substitutionsmuster *im Mittel* mit ihrer Häufigkeitsrangfolge in der betrachteten Sprache übereinstimmt. SBSTSTS geht daher sowohl bei repräsentativ als auch bei zufällig zusammengestellten Lernkorpora davon aus, daß sich an der Häufigkeitsrangfolge der Substitutionsmuster der Grad ihrer Regelhaftigkeit ablesen läßt. Selbstverständlich muß dabei in Kauf genommen werden, daß bei zufällig zusammengestellten Korpora der Grad der Regelhaftigkeit zuweilen falsch eingeschätzt wird und daher regelhaftere Substitutionsmuster mit spezielleren, kontextsensitiveren Anweisungen beschrieben werden als seltenere Substitutionsmuster. Es besteht also die Gefahr der Überdiskriminierung bei den regelhafteren und die der Übergeneralisierung bei den selteneren Substitutionsmustern.

Wie werden nun die Kontextbedingungen zu den häufigeren und den selteneren Substitutionsmustern konstruiert? Nehmen wir an, es lägen die 3 Substitutionsmuster

1. $X_1 \rightarrow Y_1$
2. $X_1 \rightarrow Y_2$
3. $X_2 \rightarrow Y_3$

in den Analyserecords von WORK2 vor, wobei X_2 echter linker Teilstring von X_1 ist. Das erste Muster sei das häufigste, das zweite trete etwas seltener auf, und das dritte sei das selten-

ste. Um die Forderungen nach schwacher Adäquatheit und nach Redundanzfreiheit der Anweisungsmenge zu erfüllen (vgl. die Forderungen 1 u. 5 in Abschnitt 3.2),

- muß mit den einzuführenden Kontextbedingungen erreicht werden, daß die 3 Anweisungen zu jedem Quellstring des Lernkorpus, der einem der 3 Substitutionsmuster angehört, den korrekten Zielstring erzeugen (schwache Adäquatheit),
- und die Kontextbedingungen müssen so kurz wie möglich formuliert werden (Redundanzfreiheit).

SBSTSTS geht daher folgendermaßen vor: Zum häufigsten Substitutionsmuster (1) wird die allgemeinste Anweisung konstruiert, die möglich ist, nämlich eine kontextfreie Anweisung. Zu 1 erhalten wir also die Substitutionsanweisung

$$X_1 \rightarrow Y_1 / \# __ .$$

Zu dem selteneren, zweiten Substitutionsmuster wird eine etwas speziellere, kontextsensitivere Substitutionsanweisung konstruiert. Die Kontextbedingung muß dabei so formuliert werden, daß die Anweisung auf keinen der (im Lernkorpus auftretenden) am ersten Substitutionsmuster beteiligten Quellstrings anwendbar ist. Um dies zu erreichen, wird zu jedem Quellstring des Lernkorpus, der am zweiten Substitutionsmuster beteiligt ist, der kürzeste linke Teilstring bestimmt, in dem der Quellstring mit keinem derjenigen Quellstrings übereinstimmt, die am ersten Substitutionsmuster beteiligt sind. (Die Quellstrings entnimmt SBSTSTS den Analyserecords in WORK2. Zur Bestimmung des kürzesten linken Teilstrings wird die Prozedur LLTMTCH (vgl. Abschnitt 5.12) verwendet.) Dieser linke Teilstring beginnt stets mit X_1. Den Kontextstring Z_1 erhält man aus dem Teilstring durch Tilgung von X_1. Ergeben sich auf diese Weise aus den an Substitutionsmuster 2 beteiligten Quellstrings mehrere identische Kontextstrings, so wird natürlich nur ein Vertreter dieser identischen Kontextstrings in die Kontextbedingung der Substitutionsanweisung aufgenommen. Ungleiche Kontextstrings $Z_1, \ldots, Z_m$ werden in der Kontextbedingung mit dem logischen 'oder' (= '|') verknüpft und in runde Klammern eingeschlossen. Zum zweiten Substitutionsmuster ergäbe sich also eine Anweisung mit einer etwas

spezielleren Kontextbedingung als zum ersten Muster:

$$X_1 \rightarrow Y_2 / \#__ (Z_1 | \ldots | Z_m)$$

Die Substitutionsanweisung mit der speziellsten Kontextbedingung wird schließlich zum seltensten, dem dritten Substitutionsmuster gebildet. Die Kontextstrings werden dabei auf die gleiche Art ermittelt wie zum zweiten Substitutionsmuster. Sie werden jedoch dadurch spezieller als die von Substitutionsanweisung 2, daß die am dritten Substitutionsmuster beteiligten Quellstrings nicht nur mit den Quellstrings des ersten, sondern auch mit denen des zweiten Substitutionsmusters verglichen werden. Um dabei aus dem kürzesten linken Teilstring, in dem ein Quellstring des dritten Substitutionsmusters mit keinem Quellstring des ersten und zweiten Substitutionsmusters übereinstimmt, einen Kontextstring zu gewinnen, muß natürlich - i. U. zum Vorgehen bei der Gewinnung der zweiten Anweisung - X_2 am Anfang des Teilstrings getilgt werden, und nicht etwa X_1. Wir erhalten damit eine dritte Substitutionsanweisung

$$X_2 \rightarrow Y_3 / \#__ (Z_1' | \ldots | Z_n'),$$

die - was sich an der hier verwendeten Notation nicht ablesen läßt - eine noch speziellere Kontextbedingung besitzt als die zum 2. Substitutionsmuster.

SBSTSTS muß nun noch die *Anwendungsreihenfolge* der generierten Substitutionsanweisungen festlegen, und zwar so, daß jede Substitutionsanweisung potentiell zur Anwendung kommen kann. Wie wir in Abschnitt 3.1 gesehen haben, genügt es dabei, die präfixorientierten Substitutionsanweisungen so anzuordnen, daß eine Anweisung, deren Konkatenationen von Substituendum und Kontextstrings linke Teilstrings der Konkatenationen von Substituendum und Kontextstrings einer anderen Anweisung sind, stets *hinter* dieser anderen Anweisung in der Menge steht. Für 2 Anweisungen, deren Konkatenationen *nicht* Teilstrings voneinander sind, stellt sich die Frage der relativen Reihenfolge nicht. Ihre Reihenfolge kann SBSTSTS beliebig gestalten. Für die 3 Beispielanweisungen ergibt sich die Reihenfolge:

$X_2 \rightarrow Y_3/\#__ (Z_1'| \ldots |Z_n')$

$X_1 \rightarrow Y_2/\#__ (Z_1| \ldots |Z_m)$

$X_1 \rightarrow Y_1/\#__$

Die ermittelten Substitutionsanweisungen speichert SBSTSTS zusammen mit der Anzahl der Quellstrings des Lernkorpus, auf die sie jeweils anwendbar sind, im Strukturarray STMT ab. Die Angabe zur Anzahl der Quellstrings wird später noch von der Prozedur STMTOUT (vgl. Abschnitt 5.5) benötigt.

Der bisher rein theoretisch dargestellte Prozess der Gewinnung präfixorientierter Substitutionsanweisungen ausgehend von WORK2 soll nun am Beispiel der Negation englischer Adjektive mit den Allomorphen des Morphems {in} veranschaulicht werden. Es sei angenommen, daß DISCOV bei Vorgabe des folgenden Lernkorpus aktiviert wird:

```
('accessible', 'inaccessible')
('cautious',   'incautious')
('variable',   'invariable')
('mature',     'immature')
('perfect',    'imperfect')
('regular',    'irregular')
('legible',    'illegible')
```

Dann erstellt PREPCRP zu diesem Lernkorpus die in Abbildung 5.2 wiedergegebenen Analyserecords, die es in der Datei WORK2 ablegt. Auf der Basis dieser Analyserecords erlernt SBSTSTS die Substitutionsanweisungen (Auf das ursprüngliche Lernkorpus wird nicht mehr zugegriffen). Für die Erlernung der präfixorientierten Substitutionsanweisungen sind allein die linken 3 Spalten der Analyserecords relevant. Die rechten 3 Spalten sind Grundlage für die Erlernung der suffixorientierten Substitutionsanweisungen.

Zur *Erstellung der präfixorientierten Substitutionsanweisungen* ermittelt SBSTSTS zunächst die Häufigkeitsrangfolge der in den 2 linken Spalten notierten Substitutionsmuster. Es ergibt sich (vgl. Abb. 5.3):

Substituendum der zu erstellenden präfixorientierten Substitutionsanweisung	Substituens der zu erstellenden präfixorientierten Substitutionsanweisung	Quellstring	Substituendum der zu erstellenden suffixorientierten Substitutionsanweisung (invertiert)	Substituens der zu erstellenden suffixorientierten Substitutionsanweisung (invertiert)	Quellstring (invertiert)
''	'in'	'accessible'	''	''	'elbissecca'
''	'in'	'cautious'	''	''	'suoituac'
''	'in'	'variable'	''	''	'elbairav'
''	'im'	'mature'	''	''	'erutam'
''	'im'	'perfect'	''	''	'tcefrep'
''	'ir'	'regular'	''	''	'raluger'
''	'il'	'legible'	''	''	'elbigel'

Abb. 5.2

Substitutionsmuster	Abs. Häufigkeit
'' → 'in'	3
'' → 'im'	2
'' → 'il'	1
'' → 'ir'	1

Abb. 5.3

Da 4 *verschiedene* Substitutionsmuster mit *identischen* Substituenda vorliegen, müssen für 3 dieser Substitutionsmuster kontext*sensitive* Substitutionsanweisungen konstruiert werden. Zunächst wird zum häufigsten Substitutionsmuster '' → 'in' die allgemeinste, weil kontext*freie* Substitutionsanweisung

'' → 'in'/# __

gebildet. Zum zweithäufigsten Bildungsmuster '' → 'im' muß eine kontext*sensitive* Substitutionsanweisung konstruiert werden, die zur Gewährleistung der schwachen Adäquatheit auf keinen der Quellstrings zum häufigsten Substitutionsmuster anwendbar sein darf. Die Kontextbedingung für diese Anweisung wird gebildet, indem zunächst zu jedem Quellstring des Lernkorpus, der am Muster '' → 'im' beteiligt ist, der kürzeste linke Teilstring ermittelt wird, in dem er mit keinem der Quellstrings zum Muster '' → 'in' übereinstimmt: Am Bildungsmuster '' → 'im' sind die Quellstrings 'mature' und 'perfect' beteiligt, am Bildungsmuster '' → 'in' die Quellstrings 'accessible', 'cautious' und 'variable'. All diese Quellstrings entnimmt SBSTSTS den Analyserecords. Kürzester linker Teilstring, in dem 'mature' mit keinem der 3 Quellstrings 'accessible', 'cautious' und 'variable' übereinstimmt, ist 'm'. Für 'perfect' ist es 'p'. Diese Teilstrings werden nun links um das Substituendum verkürzt, um die Kontextstrings für die zu konstruierende Substitutionsanweisung zum Muster '' → 'im' zu erhalten. Da das Substituendum hier der Nullstring ist, ergibt sich durch diese Verkürzung keine Veränderung der Teilstrings und SBSTSTS erhält die Kontextstrings 'm' und 'p'. Es ergibt sich somit für das Substitutionsmuster '' → 'im' die Substitutionsanweisung

'' → 'im'/# __ ('m'|'p'),

die SBSTSTS wegen ihrer Kontextsensitivität vor

'' → 'in'/# __

anordnet, so daß bei einer späteren Anwendung jede der 2 Anweisungen potentiell zum Zuge kommen kann. (Bei einer umgekehrten Anordnung käme '' → 'in'/# __ immer und '' → 'im'/# __ ('m'|'p') niemals zum Zuge, da '' → 'in'/# __ in allen Umgebungen anwendbar ist, in denen auch '' → 'im'/# __ ('m'|'p') anwendbar wäre.)

Zu den beiden seltensten Substitutionsmustern '' → 'il' und '' → 'ir' konstruiert SBSTSTS die Substitutionsanweisungen

'' → 'il'/# __'l'

und

'' → 'ir'/# __'r',

und zwar analog zur Vorgehensweise beim Muster '' → 'im'. Allerdings muß bei der Ermittlung des Kontextstrings 'l' der Quellstring 'legible' nicht nur mit 'accessible', 'cautious' und 'variable' verglichen werden, sondern auch mit 'mature' und 'perfect'. Dies ist zur Gewährleistung der schwachen Adäquatheit erforderlich. Entsprechend muß bei der Konstruktion des Kontextstrings zum Muster '' → 'ir' der diesem Muster angehörende Quellstring 'regular' zusätzlich mit 'legible' verglichen werden. Im vorliegenden Beispiel ändern diese zusätzlichen Vergleiche mit 'mature', 'perfect' und 'legible' nichts an den schließlich zustande kommenden Kontextstrings 'l' und 'r'. Diese Kontextstrings wären auch beim alleinigen Vergleich mit 'accessible', 'cautious' und 'variable' zustande gekommen. Dies ist SBSTSTS aber nicht von vornherein bekannt. Theoretisch hätte der zusätzliche Vergleich auch längere Kontextstrings erforderlich machen können. Bei anderen morphologischen Ableitungsbeziehungen tritt dieser - hier noch theoretische - Fall auch ein.

Zum vorgegebenen Lernkorpus konstruiert SBSTSTS also schließlich die folgende geordnete Menge präfixorientierter Substitutionsanweisungen:

'' → 'il'/#__'l'
'' → 'ir'/#__'r'
'' → 'im'/#__ ('m'|'p')
'' → 'in'/#__

Diese Menge speichert SBSTSTS zusammen mit der Anzahl der Quellstrings des Lernkorpus, auf die die Anweisungen anwendbar sind, im Array STMT ab.

Die *suffix*orientierten Substitutionsanweisungen erlernt SBSTSTS auf der Grundlage der rechten 3 Spalten der Analyserecords. Die Lernstrategie ist identisch mit der für die präfixorientierten Substitutionsanweisungen, bis auf den Unterschied, daß mit invertierten Zeichenketten gearbeitet wird. Da im vorliegenden Beispiel alle 7 Analyserecords in den Spalten 4 und 5 dasselbe Substitutionsmuster '' → '' aufweisen, ist die Aufstellung von Kontextbedingungen nicht erforderlich. SBSTSTS konstruiert also nur eine suffixorientierte Substitutionsanweisung. es handelt sich um die kontextfreie Anweisung

'' → ''/__#.

Diese Anweisung fordert: 'Ersetze den Nullstring durch den Nullstring, wenn er am Ende eines Quellstrings steht!' Da jede Zeichenkette mit dem Nullstring endet, ist diese Anweisung auch auf jeden Quellstring anwendbar. Sie verändert das Ende des Quellstrings jedoch überhaupt nicht. Es stellt sich daher die berechtigte Frage: Wofür braucht man diese Anweisung? Der Lernalgorithmus erzeugt diese Anweisung, damit später der Anwendungsalgorithmus bei der Anwendung von Derivationsanweisungen feststellen kann, wann *keine* der erlernten Anweisungen auf einen vorgegebenen Quellstring anwendbar ist. Würde eine solche kontextfreie Anweisung, die nichts verändert, weggelassen, so müßte für die Anwendung von Derivationsanweisungen folgende Metaregel eingeführt werden: 'Ist keine der explizit angegebenen Derivationsanweisungen auf einen Quellstring anwendbar, so gilt die implizite Derivationsanweisung: Verändere nichts!' Diese Metaregel würde auch dann gelten, wenn das Lernkorpus gar nicht die Erlernung einer kontextfreien Anweisung, welche nichts verändert,

erlaubt. Unter diesen Umständen ließe sich nicht mehr feststellen, wann keine der insgesamt vorliegenden, d. h. keine der expliziten und der impliziten Anweisungen anwendbar ist.

Wir hatten oben die Voraussetzungen angeführt, unter denen die Einführung von Kontextbedingungen in Substitutionsanweisungen erforderlich wird. Daran anschließend hatten wir die Methotik skizziert, mit der SBSTSTS Kontextbedingungen aufstellt. Nun soll darauf eingegangen werden, wann die Aufstellung von Kontextbedingungen überflüssig ist. Ex negativo formuliert liegen die Voraussetzungen, die die Einführung von Kontextbedingungen überflüssig machen, dann vor, wenn die Voraussetzungen, unter denen ihre Einführung erforderlich ist, *nicht* vorliegen. Positiv formuliert: Für ein in den Analyserecords von WORK2 auftretendes präfix-, bzw. suffixorientiertes Substitutionsmuster wird keine Kontextbedingung eingeführt,

- wenn das Substituendum dieses Substitutionsmusters nicht linker, bzw. rechter Teilstring eines anderen präfix-, bzw. suffixorientierten Substitutionsmusters ist, das in den Analyserecords von WORK2 auftritt,
- *und* wenn auch kein anderes Substituendum existiert, das zu dem betreffenden Substituendum linker, bzw. rechter Teilstring ist.

Dies sei an einem Beispiel für die Konstruktion suffixorientierter Substitutionsanweisungen illustriert. Nehmen wir an, daß folgendes (linguistisch nicht repräsentative) Lernkorpus für die Erlernung von Anweisungen vorgegeben wird, mit denen sich aus desubstantivischen deutschen Adjektiven die entsprechenden Substantive gewinnen lassen:

```
('schiffbar',          'Schiff')
('angriffsfähig',      'Angriff')
('entwicklungsfähig', 'Entwicklung')
('beschlussfähig',     'Beschluss')
('wundersam',          'Wunder')
('vorbildlich',        'Vorbild')
```

Der Inhalt der für die Konstruktion suffixorientierter Substitutionsanweisungen relevanten letzten 3 Spalten der Analyserecords, die PREPCRP zu diesem Korpus liefert, ist (hier *nicht* invertiert wiedergegeben):

'bar'	''	'schiffbar'
'sfähig'	''	'angriffsfähig'
'sfähig'	''	'entwicklungsfähig'
'fähig'	''	'beschlussfähig'
'sam'	''	'wundersam'
'lich'	''	'vorbildlich'

Aus den Substitutionsmustern

'bar' → ''

'sam' → ''

'lich' → ''

können ohne Beeinträchtigung der schwachen Adäquatheit unmittelbar die kontextfreien suffixorientierten Substitutionsanweisungen

'bar' → ''/__#

'sam' → ''/__#

'lich' → ''/__#

gewonnen werden, da keines der 3 auftretenden Substituenda zu einem anderen in den Analyserecords auftretenden Substituendum rechter Teilstring ist oder umgekehrt. Die relative Reihenfolge dieser 3 Anweisungen untereinander und zu den noch zu konstruierenden suffixorientierten Substitutionsanweisungen ist beliebig.

Die beiden verbleibenden Substitutionsmuster

'sfähig' → ''

'fähig' → ''

machen dagegen die Einführung einer Kontextbedingung erforderlich da 'fähig' rechter Teilstring von 'sfähig' ist. Die Einführung der Kontextbedingung erfolgt nach der oben beschriebenen Methode. Das häufiger im Lernkorpus auftretende Substitutionsmuster

'sfähig' → ''

wird also in die kontextfreie Substitutionsanweisung

'sfähig' → ''/__#

verwandelt, während zu dem selteneren Substitutionsmuster

'fähig' → ''

die kontextsensitive Substitutionsanweisung

'fähig' → ''/'ss'__#

erzeugt und in der Anweisungsmenge *vor*

'sfähig' → ''/__#

angeordnet wird. Wir erhalten also schließlich folgende geordnete Menge suffixorientierter Substitutionsanweisungen:

'bar' → ''/__#
'sam' → ''/__#
'lich' → ''/__#
'fähig' → ''/'ss'__#
'sfähig' → ''/__#

PREPSTS

Die letzte der 4 internen Prozeduren, die innerhalb von DISCOV durchlaufen werden, ist PREPSTS. PREPSTS bereitet die zuvor erlernten und im Strukturarray STMT abgespeicherten Derivationsanweisungen für die Weiterverarbeitung durch STMTOUT und TODISK sowie durch die beiden anderen Hauptkomponenten von PRISM auf. Die Aufbereitung besteht lediglich darin, daß die präfix- und die suffixorientierten Substitutionsanweisungen, die SBSTSTS in den Zellen des Arrays STMT in *inverser* Anwendungsreihenfolge ablegt, in die richtige Reihenfolge gebracht werden.

Zu Ende von DISCOV liegen alle erlernten Derivationsanweisungen im Array STMT vor. Und zwar stehen in den Zellen des Arrays zunächst die präfixorientierten Blockierungsanweisungen, dann die suffixorientierten Blockierungsanweisungen, dann die präfixorientierten Substitutionsanweisungen und schließlich die suffixorientierten Substitutionsanweisungen. Die Anweisungen der letzten beiden Gruppen folgen jeweils in ihrer Anwendungsreihenfolge aufeinander. Innerhalb der Gruppen der präfixorientierten und der suffixorientierten Blockierungsanweisungen ist die Anwendungsreihenfolge jeweils beliebig.

Falls bei der Ausführung von DISCOV ein Fehler auftritt, z. B. bei Sortierungen der Dateien WORK1 und WORK2 mittels des System-Hilfsprogramms SyncSort, druckt DISCOV eine entsprechende

Fehlermeldung aus. Anschließend gibt es die Kontrolle an MONITOR zurück, das aufgrund eines dabei von DISCOV übergebenen Fehlerkodes den Programmablauf beendet und die weiteren Prozeduren der Lernkomponente (STMTOUT und TODSET) nicht mehr aktiviert. Derivationsanweisungen werden in einem solchen Fall nicht erlernt.

Es wurde noch nicht erwähnt, daß DISCOV ausschließlich mit *internen* Derivationsanweisungen arbeitet und daß die Derivationsanweisungen auch im Array STMT in ihrer internen Repräsentation vorliegen. Dennoch haben wir in der obigen Darstellung der Arbeitsweise von DISCOV *externe* Derivationsanweisungen verwendet, um die Verständlichkeit unserer Ausführungen nicht zu sehr zu beeinträchtigen.

5.5 STMTOUT

Aufrufe

Aufrufende Prozedur: MONITOR.
Aufgerufene Prozedur: INVERT.

Eingabedaten

- Steueroption von der Prozedur MONITOR.
- Interne Derivationsanweisungen mit Häufigkeitsangaben aus dem Strukturarray STMT.

Ausgabedaten

Externe Derivationsanweisungen ggf. mit Häufigkeitsangaben auf den Schirm oder den Drucker.

Funktion und Arbeitsweise

Die Prozedur STMTOUT wird von MONITOR im Rahmen der Lernkomponente aktiviert, falls eine Steueroption zum Ausdrucken erlernter Derivationsanweisungen vorliegt. In diesem Fall wird sie aufgerufen, nachdem DISCOV Derivationsanweisungen erlernt und im Strukturarray STMT abgespeichert hat.

Weiterhin wird STMTOUT aktiviert bei der Ausführung der Komponente zum Ausdrucken von Derivationsanweisungen, die bei einem früheren Lernlauf in der Wissensbasis abgelegt wurden. Zuvor wird

dann allerdings die Prozedur FRODSET aufgerufen, die die auszudruckenden Derivationsanweisungen aus der Wissensbasis in das Strukturarray STMT einliest.

STMTOUT dient zum Ausdrucken von Derivationsanweisungen. Die auszudruckenden Anweisungen werden dazu an STMTOUT in ihrer internen Repräsentation im Strukturarray STMT übergeben. STMTOUT greift auf das Array zu, transformiert die Anweisungen in ihre externe Repräsentation und gibt sie in der externen Repräsentation auf dem Schirm oder dem Drucker aus. Da die Impedienten, Substituenda, Substituenten und Kontextstrings der suffixorientierten Anweisungen in der internen Repräsentation invertiert vorliegen, ruft STMTOUT bei der Transformation der suffixorientierten Anweisungen die Prozedur INVERT auf (vgl. Abschnitt 5.11), die diese invertierten Zeichenketten dann durch nochmalige Inversion in ihre nicht-invertierte Form bringt. Mit den Derivationsanweisungen gibt STMTOUT die Anzahl der Anweisungen aus sowie die Anzahl der Wortpaare des Lernkorpus, aus dem die Anweisungen gewonnen wurden. Die Anzahl der Wortpaare berechnet STMTOUT aus den Häufigkeitsangaben, die den Anweisungen im Array STMT beigegeben sind. Über eine Option, die MONITOR an STMTOUT übergibt, kann der Benutzer schließlich noch steuern, ob zu jeder externen Derivationsanweisung auch die absolute Häufigkeit derjenigen Quellstrings des Lernkorpus, auf die die Anweisung anwendbar ist, ausgegeben wird. Zusammen mit der absoluten Häufigkeit wird dann auch der prozentuale Anteil dieser Häufigkeit an der Gesamtzahl der Quellstrings des Lernkorpus ausgegeben.

Die Prozedur STMTOUT wurde im wesentlichen aus folgenden Gründen eingeführt:

- Falls die Ergebnisse der Anwendungskomponente linguistisch nicht zufriedenstellend sind, sollen die ausgegebenen Derivationsanweisungen dem Benutzer ermöglichen, die überdiskriminierenden und die übergeneralisierenden Anweisungen ausfindig zu machen. Davon ausgehend kann der Benutzer dann gezielt das Lernkorpus verbessern und in einem neuen Lernlauf eine bessere Menge von Derivationsanweisungen erstellen lassen.

- Wird PRISM als Hilfsmittel bei der Erstellung einer Grammatik der Derivationsmorphologie einer Sprache verwendet oder zur Messung der Regelhaftigkeit einer morphologischen Ableitungsbeziehung auf graphischer Ebene, so ist es natürlich notwendig, die erlernten Derivationsanweisungen mit der Angabe ihrer Anwendungshäufigkeit für das Lernkorpus auszugeben. Erst anhand dieser Angabe kann der Linguist entscheiden, inwieweit sich die Derivationsmorphologie auf graphischer Ebene mit Regeln beschreiben läßt. Diejenigen Derivationsanweisungen mit der größeren Anwendungshäufigkeit dürften eher Regeln im linguistischen Sinne repräsentieren, die auch in die Grammatik aufgenommen werden sollten. Dagegen dürften Derivationsanweisungen mit geringeren Anwendungshäufigkeiten eher Ausnahmen beschreiben. Anstelle dieser Anweisungen sollten dann in der Grammatik entsprechende Listen von Ausnahmen erscheinen.

5.6 TODSET

Aufrufe

Aufrufende Prozedur: MONITOR.

Eingabedaten

Interne Derivationsanweisungen mit Häufigkeitsangaben aus dem Strukturarray STMT.

Ausgabedaten

- Interne Derivationsanweisungen mit Häufigkeitsangaben in die Wissensbasis auf externem Datenträger.
- Informationsmeldung auf den Schirm oder den Drucker.

Funktion und Arbeitsweise

TODSET ist die letzte Prozedur, die MONITOR im Rahmen der Lernkomponente aktiviert. TODSET speichert die erlernten Derivationsanweisungen, die es aus dem Strukturarray STMT von DISCOV erhält, langfristig in der Wissensbasis auf einem externen Datenträger ab. Die Anweisungen können dann später mit der Prozedur FRODSET (vgl. Abschnitt 5.7) jederzeit wieder aus der Wissens-

basis eingelesen werden, wenn sie vom Anwendungsalgorithmus oder dem Algorithmus zum Ausdrucken von Derivationsanweisungen benötigt werden.

Abschließend gibt TODSET eine Meldung auf den Schirm oder den Drucker aus, die darüber informiert, daß die erlernten Derivationsanweisungen in der Wissensbasis abgespeichert wurden.

5.7 FRODSET

Aufrufe
Aufrufende Prozedur: MONITOR.

Eingabedaten
Interne Derivationsanweisungen mit Häufigkeitsangaben aus der Wissensbasis von externem Datenträger.

Ausgabedaten
- Interne Derivationsanweisungen mit Häufigkeitsangaben in das Strukturarray STMT.
- Ggf. eine Fehlermeldung auf den Schirm oder den Drucker.
- Ggf. ein Fehlerkode an MONITOR.

Funktion und Arbeitsweise
FRODSET wird von MONITOR zu Anfang des Anwendungsalgorithmus oder zu Anfang des Algorithmus zum Ausdrucken früher erlernter Derivationsanweisungen aufgerufen. FRODSET liest aus der Wissensbasis die Derivationsanweisungen mit ihren Häufigkeitsangaben in das Strukturarray STMT ein, in dem sie dann für die Verarbeitung durch den jeweiligen Algorithmus zur Verfügung stehen.

Falls beim Einlesen der Derivationsanweisungen aus der Wissensbasis ein Fehler auftritt, druckt FRODSET eine entsprechende Fehlermeldung aus und gibt an MONITOR einen Fehlerkode zurück, der bewirkt, daß MONITOR den Programmablauf beendet und die übrigen Prozeduren des jeweils aktivierten Algorithmus nicht mehr aufruft.

5.8 APPLY

Aufrufe

Aufrufende Prozedur: MONITOR.
Aufgerufene Prozedur: DERIVE.

Eingabedaten

Quellstrings des vom Benutzer vorgegebenen Anwendungskorpus, das auf einem externen Datenträger vorliegt.

Ausgabedaten

Quellstrings mit Zielstrings oder Fehlermeldungen in das Anwendungskorpus auf externem Datenträger.

Funktion und Arbeitsweise

APPLY wird von MONITOR als zweite Prozedur des Anwendungsalgorithmus aktiviert, nachdem FRODSET die anzuwendenden Derivationsanweisungen aus der Wissensbasis in das Strukturarray STMT eingelesen hat.

APPLY liest die Quellstrings des vom Benutzer vorgegebenen Anwendungskorpus ein, das auf einem externen Datenträger vorliegt. Es ruft für jeden Quellstring die Funktionsprozedur DERIVE mit dem Quellstring als Argument auf. DERIVE wendet dann die im Array STMT vorliegenden Derivationsanweisungen auf den Quellstring an und erzeugt dadurch einen Zielstring, der zu dem Quellstring in der durch die Derivationsanweisungen beschriebenen morphologischen Ableitungsbeziehung steht. Falls bei der Anwendung der Anweisungen ein Fehler auftritt, erzeugt DERIVE anstatt des Zielstrings eine entsprechende Fehlermeldung. Den erzeugten Zielstring oder ggf. an seiner Stelle die Fehlermeldung gibt DERIVE am Ende seiner Ausführung an APPLY zurück. APPLY schreibt darauf den Quellstring mit dem erzeugten Zielstring oder mit der Fehlermeldung in das Anwendungskorpus und geht dann zur Bearbeitung des nächsten Quellstrings im Anwendungskorpus über. Auf diese Weise läßt APPLY durch DERIVE zu allen Quellstrings des Anwendungskorpus die entsprechenden Zielstrings (bzw. Fehlermeldungen) erzeugen und gibt die Ergebnisse in das Anwendungskorpus auf dem externen Datenträger aus, wo sie dann

zur maschinellen Weiterverarbeitung mit anderen Programmen zur Verfügung stehen.

5.9 DERIVE

Aufrufe

Aufrufende Prozedur: APPLY.
Aufgerufene Prozedur: INVERT.

Eingabedaten

- Quellstrings von der Prozedur APPLY.
- Interne Derivationsanweisungen aus dem Strukturarray STMT.

Ausgabedaten

Zielstrings oder Fehlermeldungen an die Prozedur APPLY.

Funktion und Arbeitsweise

Die Funktionsprozedur DERIVE wird im Rahmen der Anwendungskomponente von der Prozedur APPLY aktiviert. Jedesmal wenn APPLY einen Quellstring aus dem Anwendungskorpus eingelesen hat, ruft es DERIVE mit diesem Quellstring als Argument auf. DERIVE greift dann auf die internen Derivationsanweisungen im Array STMT zu, in das sie zu Beginn der Anwendungskomponente von der Prozedur FRODSET eingelesen wurden. DERIVE wendet die Derivationsanweisungen auf den vorliegenden Quellstring an, erzeugt dadurch den Zielstring, der zu dem Quellstring in der durch die Derivationsanweisungen beschriebenen Ableitungsbeziehung steht, und gibt den Zielstring als Resultat an APPLY zurück, das ihn zusammen mit dem Quellstring im Anwendungskorpus abspeichert.

DERIVE wendet die internen Derivationsanweisungen nach einer anderen Strategie an, als es für ihre externen Entsprechungen in Abschnitt 3.1 erläutert wurde. Die Anwendungsstrategie von DERIVE ist in ihrem Input-Output-Verhalten identisch mit der in 3.1 beschriebenen Strategie, sie ist allerdings bedeutend schneller. Da es sich dabei um ein programmtechnisches Detail handelt, das lediglich zur Senkung des Rechenzeitbedarfs von PRISM dient und das keine neuen Erkenntnisse in sich birgt, soll auf die Anwendungsstrategie von DERIVE hier nicht näher eingegangen werden

(Wer sich für die programmtechnischen Details interessiert, dem wird sicher das im Anhang abgedruckte PRISM-Quellprogramm die gewünschten Informationen liefern).

Um die suffixorientierten Blockierungs- und Substitutionsanweisungen auf einen Quellstring anwenden zu können, muß DERIVE diesen Quellstring durch die Hilfsprozedur INVERT invertieren lassen (vgl. Abschnitt 5.11), da diese Anweisungen in ihrer internen Repräsentation mit invertierten Quellstrings arbeiten.

Bei der Anwendung von Derivationsanweisungen können Fehler auftreten. Bei einem solchen Fehler produziert DERIVE stets eine Fehlermeldung und gibt diese anstatt eines Zielstrings an APPLY zurück, das dann zusammen mit dem Quellstring die Fehlermeldung im Anwendungskorpus ablegt. Im einzelnen sind folgende 4 Arten von Fehlern möglich, die alle nur dann auftreten können, wenn die Derivationsanweisungen auf solche Quellstrings angewendet werden, die nicht im Lernkorpus vorkamen:

- Der von APPLY übergebene Quellstring weist einen formalen Fehler auf. Ein solcher Fehler läge z. B. vor, wenn der Quellstring eines von den beiden in Quellstrings nicht zugelassenen EBCDIC-Zeichen mit dem hexadezimalen Wert 00 oder FF enthält.[5]
- Die vorliegenden Derivationsanweisungen sind auf den vorgegebenen Quellstring nicht anwendbar. Dieser Fall tritt genau dann ein, wenn
  - keine Blockierungsanweisung anwendbar ist
  - *und* keine präfixorientierte und/oder keine suffixorientierte Substitutionsanweisung anwendbar ist.

  Wir wollen dies an einem Beispiel erläutern: Nehmen wir an, DERIVE soll die französische weibliche Bezeichnung (= Zielstring) 'gitane' <Zigeunerin> aus der männlichen Bezeichnung (= Quellstring) 'gitan' <Zigeuner> durch Anwendung der folgenden (linguistisch nicht adäquaten) Derivationsanweisungen

5 Die Konventionen für die Notation von Quellstrings sind vollständig im "PRISM User's Guide" (Wothke 1984: 12) wiedergegeben.

```
''     → ''/# __
'eur'  → 'rice'/'t'__#
'r'    → 'se'/__#
```

erzeugen. Dann kann kein Zielstring gebildet werden, da keine der 2 suffixorientierten Substitutionsanweisungen anwendbar ist. Natürlich ist die präfixorientierte Substitutionsanweisung '' → ''/# __ anwendbar, aber weder die suffixorientierte Anweisung 'eur' → 'rice'/'t'__# noch die suffixorientierte Anweisung 'r' → 'se'/__# kann angewendet werden, da 'gitan' weder auf 'teur' noch auf 'r' endet.

- Die Substituenda der auf den vorliegenden Quellstring anwendbaren präfixorientierten Substitutionsanweisung und der anwendbaren suffixorientierten Substitutionsanweisung überschneiden sich in dem Quellstring.
  Beispiel: Angenommen, DERIVE versucht, die präfixorientierte Substitutionsanweisung 'di' → ''/# __ und die suffixorientierte Substitutionsanweisung 'ish' → ''/__# auf den englischen Quellstring 'dish' <Schüssel> anzuwenden (Die beiden Anweisungen könnten beispielsweise aus einer Anweisungsmenge stammen, die zur Abtrennung englischer Präfixe und Suffixe dient). In diesem Fall tritt der Fehler auf, daß sich die beiden Substituenda in dem Quellstring auf dem Buchstaben 'i' überschneiden. Die 2 Anweisungen können daher nicht fehlerfrei angewendet werden.
- Der durch die Anwendung der Derivationsanweisungen zustande kommende Zielstring wäre länger als 30 Zeichen. In diesem Fall kann DERIVE keinen Zielstring erzeugen, da es nur solche Zielstrings verarbeiten kann, deren Länge kleiner oder gleich 30 Zeichen ist.

5.10 LISTOUT

Aufrufe
Aufrufende Prozedur: MONITOR.

Eingabedaten
Quellstrings mit von DERIVE erzeugten Zielstrings oder Fehlermeldungen aus dem Anwendungskorpus, das auf einem externen Datenträger vorliegt.

Ausgabedaten
Quellstrings mit von DERIVE erzeugten Zielstrings oder Fehlermeldungen auf den Schirm oder den Drucker.

Funktion und Arbeitsweise
LISTOUT wird von MONITOR als letzte Prozedur der Anwendungskomponente aktiviert, falls eine Steueroption für das Ausdrucken der vorgegebenen Quellstrings mit den erzeugten Zielstrings vorliegt.

LISTOUT liest lediglich aus dem Anwendungskorpus die Quellstrings mit den durch die Anwendung von Derivationsanweisungen erzeugten Zielstrings oder möglicherweise erzeugten Fehlermeldungen ein und gibt sie übersichtlich formatiert auf den Schirm oder den Drucker aus.

5.11 INVERT

Aufrufe
Aufrufende Prozeduren: DISCOV, STMTOUT, DERIVE.

Eingabedaten
Eine Zeichenkette und eine Steuergröße von der aufrufenden Prozedur.

Ausgabedaten
Eine invertierte Zeichenkette an die aufrufende Prozedur.

Funktion und Arbeitsweise

INVERT ist eine Funktionsprozedur. Es dient zum Invertieren von Zeichenketten. Die Prozedur führt also eine relativ einfache Operation aus. Da diese Operation allerdings häufiger und an verschiedenen Stellen in PRISM benötigt wird, bot es sich an, eine selbständige Prozedur einzurichten, die von all diesen Stellen aus aktiviert werden kann. Die Aufgabe, Zeichenketten zu invertieren, stellt sich u. a. in der Prozedur DISCOV bei der Erstellung der Analyserecords der 2 Arbeitsdateien WORK1 und WORK2.

INVERT wird mit 2 Argumenten aufgerufen. Erstes Argument ist eine Zeichenkette, zweites Argument ist eine Steuergröße. Die Steuergröße informiert INVERT darüber,

- ob es die gesamte vorgegebene Zeichenkette invertieren soll
- oder ob es den längsten linken Teilstring dieser Zeichenkette, welcher kein Leerzeichen enthält, invertieren soll
- oder ob es einen linken Teilstring mit einer durch die Steuergröße angegebenen Länge invertieren soll.

INVERT führt die gewünschte Invertierung durch und gibt die invertierte Zeichenkette oder den invertierten Teilstring der Zeichenkette an die aufrufende Prozedur zurück.

5.12 LLTMTCH

Aufrufe

Aufrufende Prozedur: DISCOV.

Eingabedaten

2 Zeichenketten und eine Steuergröße von der aufrufenden Prozedur.

Ausgabedaten

Länge der längsten linken Übereinstimmung (= longest left match) der 2 eingegebenen Zeichenketten, ggf. nur bis zu einer bestimmten Position in den Zeichenketten. Die Länge wird an die aufrufende Prozedur übergeben.

Funktion und Arbeitsweise

LLTMTCH ist eine Funktionsprozedur, die im Rahmen der Lernkomponente von der Prozedur DISCOV aktiviert wird. Es führt eine im Rahmen von DISCOV häufiger benötigte einfache Operation mit Zeichenketten durch, und zwar dient es zur Bestimmung der Länge der längsten linken Übereinstimmung zweier Zeichenketten. Diese Operation ist in DISCOV erforderlich bei der Bestimmung von Impedienten und bei der Aufstellung von Kontextbedingungen.

LLTMTCH wird mit 3 Argumenten aufgerufen: Die beiden ersten Argumente sind 2 Zeichenketten, zu denen die Länge ihrer längsten linken Übereinstimmung bestimmt werden soll. Das dritte Argument ist eine Steuergröße. Sie veranlaßt LLTMTCH, die Länge der längsten linken Übereinstimmung

- entweder nur bis zu einer bestimmten, durch die Steuergröße angegebenen Position in den 2 Zeichenketten zu bestimmen
- oder sie bis maximal zum Ende der kürzeren der 2 Zeichenketten zu bestimmen.

LLTMTCH ermittelt dann die Länge der längsten linken Übereinstimmung bis zu der durch die Steuergröße festgelegten Position und gibt sie als Resultat an DISCOV zurück.

6 WORTPAARLISTEN FÜR DIE EVALUIERUNG VON PRISM

In diesem Abschnitt werden die Wortpaarlisten beschrieben, die für die Evaluierungsläufe zur 1., 2., 3. und 4. Forderung (vgl. Abschnitt 3.2) erforderlich waren. Die eigentliche Überprüfung dieser Forderungen anhand der Evaluierungsläufe werden wir im folgenden Abschnitt (7) durchführen.

Zur 5. Forderung waren keine maschinellen Evaluierungsläufe erforderlich, da ihre Erfüllung rein deduktiv überprüft werden konnte.

6.1 Gründe für die Wahl der betrachteten Ableitungsbeziehungen

Die Evaluierungsläufe wurden im wesentlichen mit Wortpaarlisten zu 2 morphologischen Ableitungsbeziehungen durchgeführt: Zum einen wurde die Motion in der französischen Gegenwartssprache betrachtet. Zum anderen wurde die Ableitungsbeziehung zwischen Verben und Nomina actionis in der deutschen Gegenwartssprache für die Evaluierungsläufe herangezogen. Ausschlaggebend für die Wahl gerade dieser beiden Ableitungsbeziehungen war ihr unterschiedlicher Grad an Regelhaftigkeit auf graphischer Ebene:

- Die Motion im Französischen ist auf graphischer Ebene äußerst regelhaft, d. h., daß relativ wenige verschiedene Bildungsmuster an dieser Ableitungsbeziehung beteiligt sind und daß sich für die Auswahl alternativer Suffixe graphische Kontextbedingungen angeben lassen. Die hohe Regelhaftigkeit dieser Ableitungsbeziehung wird auch durch verschiedene französische Grammatiken bestätigt, so z. B. von Maurice Grevisse in der Grammatik "Le bon usage" (Grevisse 1964: 170ff.) und - etwas übersichtlicher - von H. W. Klein und F. Strohmeyer in der "Französischen Sprachlehre" (Klein/Strohmeyer 1979: 99-102).

- Dagegen ist die Ableitungsbeziehung zwischen Verben und Nomina actionis im Deutschen auf graphischer Ebene bedeutend unregelmäßiger: An dieser Ableitungsbeziehung sind wesentlich mehr verschiedene Ableitungssuffixe beteiligt als an der Motion im Französischen, und die Auswahl alternativer Suffixe ist selten graphisch oder phonologisch konditioniert. Entsprechend sind auch in Grammatiken und Wortbildungslehren des Deutschen keine Regelapparate für die Ableitung von Nomina actionis aus Verben zu finden.

Wenn PRISM erwartungsgemäß arbeitet, müßte es bei den Evaluierungsläufen für diese 2 Ableitungsbeziehungen qualitativ unterschiedliche Ergebnisse produzieren, wobei die Qualität der Ergebnisse vom Grad der Regelhaftigkeit abhängt. Konkreter: Beispielsweise müßte PRISM's Anwendungskomponente bei den Evaluierungsläufen zur 2. Forderung im Mittel wesentlich bessere Ergebnisse für die Motion liefern als für die Ableitungsbeziehung zwischen Verben und Nomina actionis. Wenn Derivationsanweisungen, die zuvor ausgehend von einer Teilmenge der an der jeweiligen Ableitungsbeziehung beteiligten Wortpaare erlernt wurden, auf nicht in der Teilmenge enthaltene Quellstrings angewendet werden, so müßten bei erwartungsgemäßem Funktionieren des Systems zu wesentlich mehr an der Motion beteiligten Quellstrings die korrekten Zielstrings erzeugt werden als bei der Ableitungsbeziehung zwischen Verben und Nomina actionis. Falls allerdings die Anwendungsergebnisse zur Motion im Mittel gleich gut oder schlechter sind als die zur Ableitungsbeziehung zwischen Verben und Nomina actionis, so würde dies bedeuten, daß der Algorithmus ein wesentliches Teilziel dieser Arbeit, nämlich die maschinelle Identifizierung morphologischer Regelmäßigkeiten auf graphischer Ebene nicht erreicht hat.

Die Beschränkung der Evaluierungsläufe auf 2 Ableitungsbeziehungen löst vielleicht in einem Punkt Widerspruch aus: Strenggenommen erfordert die Überprüfung zumindest der 2., 3. und 4. Forderung die Anwendung von PRISM auf alle morphologischen Ableitungsbeziehungen in allen natürlichen Sprachen. Da dies ein nicht durchführbares Unterfangen ist, mußten wir uns auf die

exemplarische Überprüfung der 2., 3. und 4. Forderung mit 2 Ableitungsbeziehungen beschränken.

6.2 Beschreibung der zugrunde gelegten Wortpaarlisten

Dieser Abschnitt gibt eine Erläuterung von Aufbau, Inhalt und Umfang der 2 Wortpaarlisten zur Motion im Französischen und zur Ableitungsbeziehung zwischen Verben und Nomina actionis im Deutschen, die für die Evaluierungsläufe von PRISM verwendet wurden.

Die Wortpaarliste zur Motion im Französischen

Die verwendete Wortpaarliste zur Motion im Französischen umfaßt 300 Wortpaare, deren Erstkomponenten Bezeichnungen männlicher Lebewesen sind und deren Zweitkomponenten die Bezeichnungen der entsprechenden weiblichen Lebewesen sind. Die Liste enthält also beispielsweise das Wortpaar

('danseur' <Tänzer>, 'danseuse' <Tänzerin>).

Einen Abdruck der Liste findet man im Anhang (Abschnitt 10.2).

Für die Erstellung der Liste wurde ein onomasiologisches Wörterbuch des Französischen herangezogen. Es handelt sich dabei um:

Fischer, W. (1982): Französischer Wortschatz in Sachgruppen. 6. Aufl. München.

Mit Hilfe dieses Wörterbuches ließ sich die Wortpaarliste wesentlich leichter erstellen, als es mit einem Wörterbuch möglich gewesen wäre, in dem die Einträge alphabetisch geordnet sind. Denn aufgrund der Ordnung nach Sachgruppen konnten die gesuchten männlichen und weiblichen Bezeichnungen schnell ermittelt werden, indem z. B. die Abschnitte zu den Sachgruppen "Die Tierwelt", "Verwandtschaft u. Bekanntschaft" und "Die freien Berufe" auf die gesuchten Wörter durchgegangen wurden. Dabei wurden annähernd alle männlichen und weiblichen Bezeichnungen aus dem Wörterbuch für die Wortpaarliste herausgezogen. Nicht in die Wortpaarliste aufgenommen wurden insbesondere diejenigen männlichen Bezeichnungen, bei denen nicht eindeutig geklärt werden konnte, ob eine entsprechende weibliche Wortform existiert, die zu ihnen in einer morphologischen Ableitungsbeziehung steht.

Männliche Bezeichnungen, die offensichtlich keine weibliche Entsprechung besitzen, welche zu ihnen auch in einer morphologischen Ableitungsbeziehung steht, wurden ohne weibliche Entsprechung, also ohne Zielstring, in die Wortpaarliste aufgenommen. So wurde beispielsweise 'chevalier' <Ritter> ein Quellstring ohne Zielstring. Auch 'oncle' <Onkel> wurde ohne Zielstring in die Wortpaarliste aufgenommen, obwohl die weibliche Entsprechung 'tante' <Tante> existiert. 'tante' konnte nicht als Zielstring zugelassen werden, da es nicht in einer morphologischen Ableitungsbeziehung zu 'oncle' steht. Vielmehr handelt es sich hier um Wörter mit verschiedenen Stämmen. Einzige Ausnahmen von dieser Regelung bildeten die Wortpaare

('père' <Vater>, 'mère' <Mutter>),
('parrain' <Pate>, 'marraine' <Patin>).

Diese 2 Wortpaare wurden trotz Stammverschiedenheit von Quell- und Zielstring in die Wortpaarliste aufgenommen, um bei der Anwendung von PRISM auf die Motion auch die Erlernung und Anwendung von *präfix*orientierten Substitutionsanweisungen testen zu können. Dies wäre andernfalls nicht möglich gewesen, da die Motion im Französischen nur durch *Suffix*veränderungen ausgedrückt wird und sonst kaum präfixorientierte Substitutionsanweisungen konstruiert worden wären.

Die erstellte Wortpaarliste ist für die regelhaften Bildungsmuster, die an der Motion im Französischen beteiligt sind, repräsentativ. D. h.: alle regelhaften, also häufigeren Bildungsmuster sind in der Wortpaarliste mit Beispielen vertreten, und zwar jeweils mit mehr Beispielen als die unregelmäßigeren, bzw. selteneren Bildungsmuster, soweit diese überhaupt in der Liste auftreten.

Die Wortpaarliste zur Ableitungsbeziehung zwischen Verben und Nomina actionis im Deutschen

Diese Wortpaarliste besteht aus Wortpaaren zur Ableitungsbeziehung zwischen Verben und Nomina actionis im Gegenwartsdeutschen. Die Quellstrings dieser Liste sind Verben im (nicht mit 'zu' erweiterten) Infinitiv, Zielstring zu jedem Quellstring ist ein entsprechendes Nomen actionis im Nominativ Singular. Den Terminus

'Nomen actionis' verstehen wir dabei folgendermaßen:

Def. *Nomen actionis*: Nomina actionis sind alle und nur die Substantive, die
- in einer morphologischen Ableitungsbeziehung zu einem Verb stehen
- *und* eine Handlung oder einen Vorgang bezeichnen.

Unter Handlungen verstehen wir dabei alle und nur die Prozesse, die von einem belebten oder einem personifizierten Subjekt ausgeführt werden *müssen*, während Vorgänge diejenigen Prozesse sind, die unabhängig von einem solchen Subjekt ablaufen *können*. Zu den Handlungen rechnen wir neben den äußeren Handlungen, die sinnlich wahrnehmbar sind, auch innere Handlungen, die sinnlich nicht immer wahrnehmbar sind. Beispiele für Nomina actionis, die innere Handlungen bezeichnen, sind 'Freude', 'Erkenntnis', 'Ahnung'. Dagegen bezeichnen 'Vortrag', 'Geplauder' und 'Ritt' äußere Handlungen. Beispiele für Vorgangsbezeichnungen sind 'Explosion', 'Erschütterung' und 'Zerfall'.

Ausgangspunkt für die Erstellung der Wortpaarliste waren 20.262 deutsche Verbformen im Infinitiv. Diese Verbenliste wurde uns freundlicherweise von den Mitarbeitern des Projektes "Aufbau einer kumulierten Wortdatenbank des Deutschen" zur Verfügung gestellt, das in den Jahren 1982-1984 am Institut für Kommunikationsforschung und Phonetik der Universität Bonn durchgeführt wurde. Die Liste ist ursprünglich Bestandteil eines maschinellen Lexikons der deutschen Verben, das in dem Projekt erstellt wurde.

Für die Erstellung der Wortpaarliste wurde die Verbenliste folgendermaßen aufbereitet:

- Verben, die nach unserer subjektiven Einschätzung nicht dem allgemeinen Sprachgebrauch der deutschen Hochsprache der Gegenwart angehören, die also Dialekten oder Geheimsprachen angehören oder die veraltet sind, wurden gelöscht, falls sie auch in mindestens einem der folgenden 4 herangezogenen Wörterbücher als dialektal, geheimsprachlich oder veraltet markiert sind:

Duden (1976ff.): Das große Wörterbuch der deutschen Sprache in sechs Bänden. 6 Bde. Mannheim/Wien/Zürich.

Mackensen, L. (1962): Deutsches Wörterbuch. 4. verb. u. erw. Aufl. Baden-Baden.

Mater, E. (1966): Deutsche Verben. Bd. 1. Alphabetisches Gesamtverzeichnis. Leipzig.

Wahrig, G. u. a. (Hrsg.) (1977): Deutsches Wörterbuch. Überarb. Aufl. Gütersloh/Berlin.

Falls die Verben in keinem der 4 Wörterbücher als dialektal, geheimsprachlich oder veraltet markiert waren, verblieben sie in der Liste.

- Einträge, die keine Verben oder keine reinen (nicht erweiterten) Infinitive sind, wurden gelöscht. So wurden beispielsweise alle mit 'zu' erweiterten Infinitive (Beispiel: 'abzubeissen') getilgt.
- Rechtschreibfehler bei den Verben wurden korrigiert. Ergab sich durch die Korrektur ein bereits in der Liste vorhandener Eintrag, so wurde der falsch geschriebene Eintrag gelöscht.

Durch diese Aufbereitung wurde die Verbliste um 6.302 Einträge vermindert, so daß sie schließlich 13.960 Einträge umfaßte. Dieser Umfang liegt knapp unter der Anzahl von 14.560 Verben, die Erich Mater (1966: ohne Paginierung) in seinem alphabetischen Gesamtverzeichnis der deutschen Verben als der "lebenden Hoch- und Gemeinsprache" angehörig und vom "durchschnittlich gebildeten Mitglied unserer Sprachgemeinschaft" als zumindest passiv beherrscht klassifiziert.

Den Verben dieser reduzierten Liste wurden die entsprechenden Nomina actionis hinzugefügt. Als Hilfsmittel dienten dabei

- das Duden-Wörterbuch (Duden 1976ff.),
- das "Deutsche Wörterbuch" von Mackensen (1962),
- und das "Deutsche Wörterbuch" von Wahrig (1977).

Um die in Regel V (vgl. Abschnitt 3.1) geforderte Rechtseindeutigkeit der Wortpaarliste zu gewährleisten, konnte jedem Verb nur ein Nomen actionis hinzugefügt werden. Existieren also mehrere Nomina actionis, wie etwa 'Singen' (= substantivierter Infinitiv), 'Gesang', 'Singerei' und 'Gesinge' zu dem Verb 'singen', so mußte eines dieser Nomina actionis für die Aufnahme in die Wortpaarliste ausgewählt werden. Die Auswahl des zu ergänzenden Nomen actionis richtete sich nach den folgenden Kri-

terien:

1. Das Nomen actionis muß dem allgemeinen Sprachgebrauch der deutschen Hochsprache der Gegenwart angehören.

   Nomina actionis, die Dialekten oder Geheimsprachen angehören oder die veraltet sind, wurden also nicht in die Wortpaarliste aufgenommen.

2. Das Nomen actionis muß eine wertneutrale Bezeichnung einer länger andauernden (= durativen) Handlung oder eines länger andauernden Vorgangs sein.

   Nicht in die Wortpaarliste aufgenommen wurden Nomina actionis mit einer stark pejorativen oder resultativen Bedeutungskomponente sowie Nomina actionis, die einen vereinzelten Vorgang oder eine vereinzelte Handlung bezeichnen.
   Beispiele:
   - Nomina actionis mit pejorativen Konnotationen sind häufig mit den Suffixen 'erei', 'elei', 'ei', dem Präfix 'Ge' oder der Präfix-Suffix-Kombination 'Ge ... e' gebildet. Als Zielstring zu 'singen' wurde also nicht 'Singerei' oder 'Gesinge' gewählt.
   - Zu 'erkennen' existiert neben 'Erkennung', das stärker den Ablauf eines Prozesses bezeichnet, das Nomen actionis 'Erkenntnis', das eine stärker resultative Bedeutungskomkonente hat und eher das Ergebnis eines Prozesses bezeichnet. Es wurde also nicht 'Erkenntnis', sondern 'Erkennung' als Zielstring zu 'erkennen' gewählt.
   - Ein vereinzelter Vorgang oder eine vereinzelte Handlung wird häufig mit Nomina actionis bezeichnet, die mittels des Suffixes 'er' vom Verb abgeleitet sind (Beispiele: 'Seufzer', 'Hopser', 'Jauchzer'). Zu 'seufzen' wurde nicht 'Seufzer', sondern der substantivierte Infinitiv 'Seufzen' als Zielstring gewählt.

   Eine Ausnahme von Kriterium 2 bilden Zustandsverben wie 'besitzen' und 'bestehen'. Da ja zu diesen Verben keine Nomina actionis existieren, wurden die entsprechenden abgeleiteten Zustandsbezeichnungen, im Beispiel also 'Besitz' und 'Bestand', in die Liste aufgenommen.

3. Existieren zu einem Verb mehrere Nomina actionis, die die Kriterien 1 und 2 erfüllen, so wird dasjenige Nomen actionis als Zielstring gewählt, das
   - nicht substantivierter Infinitiv ist
   - *und* einen möglichst breiten Bedeutungsbereich abdeckt.

4. Ist nicht sicher, ob zu einem Verb ein Nomen actionis existiert, das nicht gleich dem substantivierten Infinitiv ist, so wird grundsätzlich der substantivierte Infinitiv als Zielstring gewählt.

Die 4 Regeln wurden mit dem Nebengedanken einer möglichen Anwendung der Verb-Nomen actionis-Liste außerhalb des Systems PRISM formuliert: Wie die Darstellung der Arbeiten zur Derivationsmorphologie von Pacak/Pratt (Abschnitt 2.1.3) und im Projekt JUDO (Abschnitt 2.1.5) gezeigt hat, ist die maschinelle Nominalisierung von Verben ein Faktor, von dem man sich in linguistisch orientierten Informationssystemen eine Erhöhung des Recall erhofft. Im Projekt JUDO ergab sich bei der Arbeit mit gegenwartsdeutschen juristischen Fachtexten, daß bei Berücksichtigung der Ableitungsbeziehung zwischen Verben und Nomina actionis

> "eine erhebliche Verbesserung der Retrievalergebnisse zu erwarten (ist; Ergänzung des Verf.), da sich Verbalabstraktum und Verb semantisch fast immer decken, was bei den anderen Derivationstypen in dieser uneingeschränkten Form nicht unbedingt gilt." (Viets 1981: 4)

Entsprechend hoch ist bei JUDO der Anteil der in den Thesaurus aufgenommenen Verb-Substantiv-Paare im Vergleich zu den Substantiv-Adjektiv- und Verb-Adjektiv-Paaren (vgl. dazu Abb. 2.6). Da unsere Verb-Nomen actionis-Liste den passiven Verbwortschatz eines durchschnittlich gebildeten deutschen Sprechers nahezu vollständig abdeckt, bot es sich an, die Nomina actionis so auszuwählen, daß sich die Liste auch für einen möglichen Einsatz in einem Informationssystem eignet. Sie könnte dort als maschinelles Lexikon für die Bereitstellung von Nomina actionis zu Verben fungieren. Entsprechend haben wir einem Verb immer dasjenige Nomen actionis hinzugefügt, von dem wir glauben, daß es im Rahmen dieser Anwendung am ehesten von Interesse ist: Dialektale, veraltete oder geheimsprachliche Bildungen sowie Nomina actionis mit pejorativer Konnotation wurden deshalb beispielsweise nicht in die Liste aufgenommen. Nomina actionis,

die dem allgemeinen Sprachgebrauch der deutschen Hochsprache der Gegenwart angehören, wurden dagegen bevorzugt, soweit sie wertneutrale Bezeichnungen länger andauernder Handlungen oder Vorgänge sind.

7 HYPOTHESENÜBERPRÜFUNG

Gegenstand dieses Abschnittes ist die Überprüfung der 5 Forderungen, die wir in Abschnitt 3.2 an den zu konstruierenden Algorithmus und die von ihm zu erzeugenden Mengen von Derivationsanweisungen gestellt hatten. Die Forderungen sollen hier nicht noch einmal im einzelnen wiedergegeben werden. Wir verweisen dafür auf Abschnitt 3.2.

1. Forderung

Die erste Forderung ist erfüllt: Der Lernalgorithmus konstruiert zu jedem Lernkorpus, das den Regeln I-V (vgl. Abschnitt 3.1) entspricht, eine Menge maschinell ausführbarer Derivationsanweisungen, die bei ihrer Ausführung durch den Anwendungsalgorithmus zu jedem Quellstring des Lernkorpus eine Zeichenkette erzeugt, die identisch ist mit dem Zielstring, welcher dem Quellstring im Lernkorpus beigegeben ist.

Begründung: Die erste Forderung wurde zunächst *induktiv* überprüft, indem vom Anwendungsalgorithmus Mengen von Derivationsanweisungen, die zuvor der Lernalgorithmus zu verschiedenen Lernkorpora erlernt hatte, auf die Quellstrings des jeweils zugrunde liegenden Lernkorpus angewandt und dadurch Zielstrings zu den Quellstrings erzeugt wurden. Als Lernkorpora wurden dabei die Wortpaarliste zur Motion, die Verb-Nomen actionis-Liste und weitere Wortpaarlisten verwendet, auf die hier nicht näher eingegangen werden soll. Die automatisch erzeugten Zielstrings wurden dann von einem Evaluierungsprogramm maschinell mit den im Lernkorpus vorgegebenen Zielstrings auf Identität verglichen. Es stellte sich heraus, daß alle maschinell erzeugten Zielstrings mit den vorgegebenen Zielstrings identisch waren. Somit konnte die 1. Forderung zumindest experimentell bestätigt werden. Die induktive Überprüfung reicht nun keineswegs aus, um feststellen

zu können, ob Forderung 1 generell erfüllt ist, da das positive Ergebnis dieser Überprüfung durchaus zufällig zustande gekommen sein kann und man möglicherweise für andere Lernkorpora negative Ergebnisse erhält. Da Forderung 1 - i. U. zu den Forderungen 2-4 - *deduktiv* und somit vollständig anhand der Funktionsweise der Lernkomponente, und zwar speziell der Prozedur DISCOV (vgl. Abschnitt 5.4), überprüft werden kann, wollen wir die deduktive Überprüfung dieser Forderung anschließen:

Es gilt zu zeigen, daß DISCOV zu jedem Lernkorpus eine Menge von Derivationsanweisungen konstruiert, die zu jedem Quellstring des Lernkorpus eine Zeichenkette erzeugt, die mit dem im Lernkorpus vorgegebenen Zielstring übereinstimmt. M. a. W.: Es gilt zu zeigen, daß DISCOV zu jedem Lernkorpus eine *schwach adäquate* Menge von Derivationsanweisungen erlernt.

Gegeben sei ein beliebiges Lernkorpus

$$
\begin{aligned}
\mathcal{L} = \{&(a_1, \ ''),\\
&(a_2, \ ''),\\
&\quad\vdots\\
&(a_{n-1}, \ ''),\\
&(a_n, \ ''),\\
&(b_1, \ c_1),\\
&(b_2, \ c_2),\\
&\quad\vdots\\
&(b_{m-1}, \ c_{m-1}),\\
&(b_m, \ c_m)\},
\end{aligned}
$$

zu dem DISCOV eine Menge von Derivationsanweisungen erlernen soll. $\mathcal{L}$ enthält n Wortpaare mit leerem Zielstring, nämlich

$(a_1, \ '')$,
$(a_2, \ '')$,

.
.
.

$(a_{n-1}, \ '')$,
$(a_n, \ '')$,

sowie m Wortpaare mit nicht-leerem Zielstring:

$(b_1, \ c_1)$,
$(b_2, \ c_2)$,

.
.
.

$(b_{m-1}, \ c_{m-1})$,
$(b_m, \ c_m)$.

1. Wir beweisen zunächst, daß DISCOV die *Blockierungsanweisungen* so konstruiert, daß sie bei ihrer späteren Anwendung für jedes a_i ($1 <= i <= n$) die Erzeugung eines Zielstrings verhindern und daß sie auf b_1, b_2, ..., b_{m-1}, b_m nicht anwendbar sind, so daß auf b_1, b_2, ..., b_{m-1}, b_m noch die Substitutionsanweisungen anzuwenden versucht werden müssen (, für die wir unter 2 beweisen, daß sie zu jedem b_j ($1 <= j <= m$) den korrekten Zielstring erzeugen).

   a) Nehmen wir zunächst an, daß $n > 0$ und $m = 0$, d. h., daß in $\mathcal{L}$ ausschließlich Paare mit leerem Zielstring vorkommen. In diesem Fall konstruiert DISCOV mit der in 5.4 dargestellten Strategie nur die Blockierungsanweisung

   $\# \ '' \neg$.

   Diese Anweisung besagt, daß zu einem Quellstring kein Zielstring zu erzeugen ist, falls der Quellstring mit dem Nullstring beginnt. Da jeder Quellstring mit dem Nullstring beginnt, verhindert diese Anweisung für jeden Quellstring aus $\mathcal{L}$ die Erzeugung eines Zielstrings.

   Damit ist bewiesen, daß DISCOV für jedes Lernkorpus,

das ausschließlich Paare mit leerem Zielstring enthält, eine schwach adäquate Menge von Derivationsanweisungen erzeugt.

b) Es sei n > 0 und m > 0, d. h., $\mathcal{L}$ enthält sowohl Paare mit leerem Zielstring als auch Paare mit nicht-leerem Zielstring. In diesem Fall konstruiert DISCOV ausgehend von $a_1, \ldots, a_n$ eine Menge präfixorientierter Blockierungsanweisungen

$\# X_1 \neg$

.
.
.

$\# X_k \neg$

und suffixorientierter Blockierungsanweisungen

$X_1' \# \neg$

.
.
.

$X_l' \# \neg$.

Wie in 5.4 beschrieben, sind die $X_1, \ldots, X_k$ kürzeste linke und die $X_1', \ldots, X_l'$ kürzeste rechte Teilstrings, in denen die a_i ($1 <= i <= n$) mit keinem der b_j ($1 <= j <= m$) übereinstimmen. Es gilt daher zunächst, daß die konstruierten Blockierungsanweisungen auf kein b_j, also auf keinen Quellstring mit nicht-leerem Zielstring anwendbar sind. Nun ist noch zu zeigen, daß zu jedem a_i unter den konstruierten Blockierungsanweisungen mindestens eine existiert, die auf a_i anwendbar ist. Es ist also zu zeigen, daß zu jedem a_i mindestens ein X_h ($1 <= h <= k$) linker oder mindestens ein X_g' ($1 <= g <= l$) rechter Teilstring ist: Die Prozedur BLCKSTS (vgl. Abschnitt 5.4) konstruiert aus den von PREPCRP gelieferten Analyserecords zu $a_1, \ldots, a_n$ die Blockierungsanweisungen. Zu jedem a_i existiert dabei ein Analyserecord, der jeweils den kürzesten linken und den kürzesten rechten Teilstring enthält, in dem a_i mit keinem

b_j übereinstimmt. Wie in 5.4 beschrieben, konstruiert BLCKSTS zu jedem Analyserecord

- *entweder* eine präfixorientierte Blockierungsanweisung mit dem kürzesten linken Teilstring als Impediens X_h
- *oder* eine suffixorientierte Blockierungsanweisung mit dem kürzesten rechten Teilstring als Impediens X_g'
- *oder* keine Blockierungsanweisung, falls bereits eine präfixorientierte Blockierungsanweisung mit dem kürzesten linken Teilstring als Impediens oder eine suffixorientierte Blockierungsanweisung mit dem kürzesten rechten Teilstring als Impediens existiert.

Somit konstruiert BLCKSTS eine Menge von Blockierungsanweisungen, von denen mindestens eine auf jedes a_i anwendbar ist, da entweder eines der präfixorientierten Impedienten X_h linker oder eines der suffixorientierten Impedienten X_g' rechter Teilstring des a_i ist.

c) Es sei n = 0, d. h., $\mathcal{L}$ enthält keinen Quellstring mit leerem Zielstring. In diesem Fall erlernt DISCOV keine Blockierungsanweisungen. Somit kann keine Blockierungsanweisung bei den möglicherweise noch in $\mathcal{L}$ auftretenden Quellstrings mit nicht-leerem Zielstring fälschlicherweise die Erzeugung eines Zielstrings verhindern.

2. Es gilt nun noch zu zeigen, daß DISCOV die *Substitutionsanweisungen* so konstruiert, daß sie zu jedem b_j ($1 <= j <= m$) einen korrekten Zielstring erzeugen. Es muß dagegen nicht mehr nachgewiesen werden, daß die Substitutionsanweisungen auf kein a_i ($1 <= i <= n$) anwendbar sind, da 1 bereits gezeigt hat, daß auf jedes a_i mindestens eine der von DISCOV erlernten Blockierungsanweisungen anwendbar ist. Wegen der partiellen Ordnung (vgl. Abschnitt 3.1) von Mengen von Derivationsanweisungen können daher die Substitutionsanweisungen für die a_i nicht zur Anwendung kommen. Die a_i können also im folgenden ignoriert werden.

a) Nehmen wir zunächst an, daß m > 0. In diesem Fall konstruiert DISCOV's interne Prozedur PREPCRP zu jedem Paar (b_j, c_j) ($1 <= j <= m$) einen Analyserecord, der neben b_j bereits die Substituenda und Substituenten der zu kon-

struierenden präfix- und suffixorientierten Substitutionsanweisungen enthält. Daß dies die korrekten Substituenda und Substituenten sind, hat bereits die Beschreibung von PREPCRP in 5.4 gezeigt. Im Anschluß an PREPCRP konstruiert SBSTSTS

- zu jedem präfixorientierten Substituendum-Substituens-Paar in den Analyserecords eine präfixorientierte Substitutionsanweisung, vorausgesetzt, es existiert nicht bereits eine präfixorientierte Substitutionsanweisung mit diesem Substituendum-Substituens-Paar.
- und zu jedem suffixorientierten Substituendum-Substituens-Paar in den Analyserecords eine suffixorientierte Substitutionsanweisung, vorausgesetzt, es existiert nicht bereits eine suffixorientierte Substitutionsanweisung mit diesem Substituendum-Substituens-Paar.

α) Existieren dabei unter den Substituendum-Substituens-Paaren *keine* präfixorientierten, deren Substituenda linke Teilstrings voneinander sind, und auch *keine* suffixorientierten, deren Substituenda rechte Teilstrings voneinander sind, so werden aus allen Paaren kontext*freie* Substitutionsanweisungen konstruiert. Da die Substituenda nicht Teilstrings voneinander sind, ist jede dieser Anweisungen dann nur auf die Quellstrings derjenigen Wortpaare anwendbar, aus denen PREPCRP das Substituendum-Substituens-Paar der Substitutionsanweisung konstruiert hatte. Außerdem ist auch auf jeden dieser Quellstrings eine Substitutionsanweisung anwendbar, da PREPCRP zu *jedem* Wortpaar ein präfix- und ein suffixorientiertes Substituendum-Substituens-Paar konstruiert hatte, zu dem SBSTSTS dann - teilweise stellvertretend für mehrere identische Substituendum-Substituens-Paare - eine präfix- und eine suffixorientierte Substitutionsanweisung aufstellte. Da PREPCRP ausschließlich korrekte Substituendum-Substituens-Paare bereitstellt, gilt für Fall α also, daß DISCOV die Substitutionsanweisungen so erstellt, daß sie zu jedem b_j einen korrekten Zielstring erzeugen, was zu beweisen war.

ß) Existieren dagegen unter den Substituendum-Substituens-Paaren präfixorientierte Substituenda, die linke Teilstrings voneinander sind, oder suffixorientierte Substituenda, die rechte Teilstrings voneinander sind, so führt SBSTSTS bei den aus diesen Paaren hervorgehenden Anweisungen Kontextbedingungen ein, die festlegen, unter welchen kontextuellen Voraussetzungen die Substitutionsanweisungen anwendbar sind. Zu zeigen ist nun daß die Kontextbedingungen so gewählt werden, daß jede dieser Anweisungen auf alle und nur die Quellstrings derjenigen Wortpaare anwendbar ist, aus denen PREPCRP das Substituendum-Substituens-Paar der Anweisung konstruiert hatte. Wir zeigen dies hier nur für präfixorientierte Substitutionsanweisungen. Für suffixorientierte Substitutionsanweisungen verläuft der Beweis analog. Es sei angenommen, daß eine Menge von präfixorientierten Substitutionsmustern

$X_1 \rightarrow Y_1$
$X_2 \rightarrow Y_2$

.
.
.

$X_l \rightarrow Y_l$
$X_{l+1} \rightarrow Y_{l+1}$

.
.
.

$X_k \rightarrow Y_k$

in den Analyserecords in WORK2 vorliegt, wobei

- für jedes X_g ($1 <= g <= l$) gelte, daß es linker Teilstring zu mindestens einem anderen X_h ($1 <= h <= l$ & $g \neq h$) ist, jedoch zu keinem X_c ($l+1 <= c <= k$),
- und für jedes X_c ($l+1 <= c <= k$) gelte, daß es zu keinem X_d ($l+1 <= d <= k$ & $c \neq d$) und zu keinem X_g ($1 <= g <= l$) linker Teilstring ist.

Mit den Mustern $X_{l+1} \rightarrow Y_{l+1}$, ..., $X_k \rightarrow Y_k$ konstruiert

SBSTSTS kontextfreie Substitutionsanweisungen, da bereits durch die Substituenda dieser Anweisungen gewährleistet ist, daß sie jeweils auf alle und nur die Quellstrings b_j aus $\mathcal{L}$ anwendbar sind, die den durch diese Anweisungen jeweils beschriebenen Substitutionsmustern angehören. Die relative Reihenfolge dieser Anweisungen untereinander und zu den Anweisungen, die aus $X_1 \rightarrow Y_1$, ..., $X_l \rightarrow Y_l$ entstehen, spielt dabei keine Rolle.

Zu den Substitutionsmustern $X_1 \rightarrow Y_1$, ..., $X_l \rightarrow Y_l$ müssen dagegen kontextsensitive Substitutionsanweisungen aufgestellt werden: Da kein X_g ($1 <= g <= l$) linker Teilstring eines X_c ($l+1 <= c <= k$) ist und umgekehrt, sind diese Anweisungen bereits ohne Kontextbedingungen auf keinen der Quellstrings aus $\mathcal{L}$ anwendbar, die den Substitutionsmustern $X_{l+1} \rightarrow Y_{l+1}$, ..., $X_k \rightarrow Y_k$ angehören. Diese Quellstrings und die Anweisungen $X_{l+1} \rightarrow Y_{l+1}/\#__$, ..., $X_k \rightarrow Y_k/\#__$ müssen daher bei der folgenden Aufstellung der Kontextbedingungen und der Festlegung der relativen Reihenfolge der aus $X_1 \rightarrow Y_1$, ..., $X_l \rightarrow Y_l$ entstehenden Anweisungen nicht mehr beachtet werden. Wir nehmen an, daß von den Substitutionsmustern $X_1 \rightarrow Y_1$, ..., $X_l \rightarrow Y_l$ diejenigen mit den niedrigeren Indices seltener oder gleich häufig implizit in $\mathcal{L}$ vertreten sind wie die mit den höheren Indices. SBSTSTS konstruiert dann zum seltensten Substitutionsmuster $X_1 \rightarrow Y_1$ die kontextsensitivste der l Substitutionsanweisungen, indem die kürzesten linken Teilstrings ermittelt werden, in denen diejenigen Quellstrings aus $\mathcal{L}$, die dem seltensten Muster angehören, mit keinem derjenigen Quellstrings übereinstimmen, die den häufigeren Mustern $X_2 \rightarrow Y_2$, ..., $X_l \rightarrow Y_l$ angehören. Diese kürzesten Nichtübereinstimmungen werden dann links jeweils um X_1 verkürzt und als Kontextbedingungen der aus $X_1 \rightarrow Y_1$ entstehenden Anweisung beigegeben, wodurch die kontextsensitive Anweisung

$$X_1 \rightarrow Y_1/\#__ \ (Z_1| \ \ldots \ |Z_p)$$

entsteht, die an den Anfang der Menge präfixorientierter Substitutionsanweisungen gesetzt wird, so daß sie später immer als erste präfixorientierte Substitutionsanweisung anzuwenden versucht wird. Durch die Kontextbedingung ist dann garantiert, daß die Anweisung auch für alle Quellstrings aus $\mathcal{L}$ zur Anwendung kommt, die dem Muster $X_1 \rightarrow Y_1$ angehören, wodurch zumindest am Anfang der Quellstrings die korrekte Veränderung für die Erzeugung des Zielstrings durchgeführt wird (Am Ende würde der Quellstring noch durch eine suffixorientierte Substitutionsanweisung verändert, worauf hier allerdings nicht näher eingegangen werden soll). Weiterhin gewährleistet die Kontextbedingung, daß die Anweisung auf keinen der Quellstrings b_j aus $\mathcal{L}$ anwendbar ist die nicht dem Substitutionsmuster $X_1 \rightarrow Y_1$ angehören. Zum nächstselteneren Substitutionsmuster $X_2 \rightarrow Y_2$ konstruiert SBSTSTS auf die gleiche Methode wie zu $X_1 \rightarrow Y_1$ eine Kontextbedingung, mit dem einzigen Unterschied, daß die kürzesten linken Nichtübereinstimmungen nur durch Vergleich mit den Quellstrings, die den nun häufigeren Substitutionsmustern $X_3 \rightarrow Y_3$, ..., $X_1 \rightarrow Y_1$ angehören, ermittelt und anstatt um X_1 links um X_2 verkürzt werden. Dadurch entsteht i. allg. eine etwas weniger kontextsensitive Substitutionsanweisung als zum ersten Substitutionsmuster, die in der Menge präfixorientierter Substitutionsanweisungen als zweite angeordnet wird. Wird versucht, die jetzt vorliegende Menge kontextsensitiver Substitutionsanweisungen auf diejenigen Quellstrings aus $\mathcal{L}$ anzuwenden, die dem zweiten Substitutionsmuster angehören, so gewährleisten die Kontextbedingungen dieser 2 Anweisungen, daß die erste Anweisung auf keinen dieser Quellstrings anwendbar ist, die zweite Anweisung dagegen auf jeden dieser Quellstrings, wodurch die korrekten Veränderungen am Anfang der Quellstrings durchgeführt werden. Aufgrund der Kontextbedingungen sind die 2 Anweisungen weiterhin auf keinen der Quellstrings b_j aus $\mathcal{L}$ anwendbar, die nicht

den Substitutionsmustern $X_1 \rightarrow Y_1$ und $X_2 \rightarrow Y_2$ angehören. Da die kontextsensitiven Substitutionsanweisungen zu den Mustern $X_3 \rightarrow Y_3$, ..., $X_{l-1} \rightarrow Y_{l-1}$ auf analoge Weise konstruiert werden wie die beiden ersten Anweisungen, gilt für sie ebenfalls, daß sie jeweils auf alle und nur diejenigen Quellstrings aus $\mathcal{L}$ anwendbar sind, die den jeweiligen Substitutionsmustern angehören. Zum häufigsten Substitutionsmuster $X_l \rightarrow Y_l$ wird schließlich die kontextfreie Anweisung

$$X_l \rightarrow Y_l / \# __$$

konstruiert und am Ende der Menge präfixorientierter Substitutionsanweisungen angeordnet, so daß für sie immer erst ein Anwendungsversuch stattfindet, wenn festgestellt worden ist, daß die davor stehenden l-1 Anweisungen auf den jeweils vorgegebenen Quellstring nicht anwendbar sind. Da diese Anweisungen jeweils auf alle und nur *die* Quellstrings b_j aus $\mathcal{L}$ anwendbar sind, die jeweils den Substitutionsmustern $X_1 \rightarrow Y_1$, ..., $X_{l-1} \rightarrow Y_{l-1}$ angehören, bleiben als einzige Quellstrings b_j aus $\mathcal{L}$, auf die all diese l-1 Anweisungen vergeblich anzuwenden versucht werden, diejenigen, die dem Muster $X_l \rightarrow Y_l$ angehören. Nur auf diese Quellstrings wird versucht, die letzte Anweisung $X_l \rightarrow Y_l / \# __$ anzuwenden. Da diese Anweisung kontextfrei ist, ist sie auf all diese Quellstrings anwendbar und führt die korrekte Veränderung am Anfang der Quellstrings durch.

Für Fall β ist somit gezeigt, daß DISCOV die präfixorientierten Substitutionsanweisungen so konstruiert, daß sie bei jedem b_j die korrekte und notwendige Veränderung für die Erzeugung des c_j durchführen.

Da die suffixorientierten Substitutionsanweisungen auf analoge Weise konstruiert werden wie die präfixorientierten Substitutionsanweisungen, gilt für sie ebenfalls, daß sie zu jedem Quellstring b_j aus $\mathcal{L}$ die korrekte graphische Veränderung durchführen, und zwar am Ende des Quellstrings.

Es gilt somit insgesamt auch für Fall β, daß DISCOV die Substitutionsanweisungen so konstruiert, daß sie zu jedem b_j einen korrekten Zielstring erzeugen, was zu beweisen war.

b) Es sei m = 0. In diesem Fall erübrigt es sich zu zeigen, daß DISCOV die Substitutionsanweisungen so konstruiert, daß sie zu jedem b_j den korrekten Zielstring erzeugen, denn das Lernkorpus enthält kein b_j. Im übrigen konstruiert DISCOV für m = 0 auch keine Substitutionsanweisungen.

2. Forderung

Zur Überprüfung der 2. Forderung wurde sowohl die 300 Paare umfassende Wortpaarliste zur Motion im Französischen als auch die 13.960 Wortpaare umfassende Verb-Nomen actionis-Liste zunächst in ein gleich großes Lern- und Anwendungskorpus geteilt, indem in das Lernkorpus jeweils die Wortpaare aus den Records mit den geraden Recordnummern der alphabetisch aufsteigend sortierten Listen aufgenommen wurden und in das Anwendungskorpus jeweils die Wortpaare aus den Records mit den ungeraden Recordnummern (Erste Recordnummer war 1). Lern- und Anwendungskorpus enthielten also

- bei der Motion im Französischen jeweils 150 Paare,
- bei der Ableitungsbeziehung zwischen Verben und Nomina actionis jeweils 6.980 Paare.

Für die zur Überprüfung der 2. Forderung erforderliche Vergrößerung der zum Lernen vorgegebenen Wortpaarlisten wurden aus den 2 Lernkorpora schrittweise größere Teil-Lernkorpora extrahiert. Insgesamt wurden aus jedem der 2 Lernkorpora jeweils 10 Teil-Lernkorpora entnommen. Bei der Wortpaarliste zur Motion war jedes Teilkorpus um 15 Paare, bei der Verb-Nomen actionis-Liste um 698 Paare größer als das jeweils nächstkleinere Teil-Lernkorpus. Da das kleinste Teil-Lernkorpus 15, bzw. 698 Paare enthielt, war also das größte Teil-Lernkorpus identisch mit dem jeweiligen gesamten Lernkorpus. Es ergaben sich somit für die Evaluierungsläufe Teil-Lernkorpora mit folgendem Umfang:

Nummer des Teil-Lern-korpus	Anzahl der Wortpaare im Teil-Lernkorpus zur Motion im Französischen	Ableitungsbeziehung zwischen Verben und Nomina actionis im Deutschen
1	15	698
2	30	1.396
3	45	2.094
4	60	2.792
5	75	3.490
6	90	4.188
7	105	4.886
8	120	5.584
9	135	6.282
10	150	6.980

Abb. 7.1

Die kleineren Teil-Lernkorpora waren jeweils Teilmengen der größeren Teil-Lernkorpora.

Für die Überprüfung der 2. Forderung ist die (aus linguistischer Sicht) *zufällige* Zusammenstellung der Teil-Lernkorpora erforderlich. Um sie zu gewährleisten, wurden die Teil-Lernkorpora mit einem Zufallsverfahren aus dem Lernkorpus extrahiert.

Die Lernkomponente von PRISM erlernte nun zunächst zum 1. Teil-Lernkorpus eine Menge von Derivationsanweisungen. Anschliessend wurden die erlernten Derivationsanweisungen von PRISM's Anwendungskomponente auf die Quellstrings des Anwendungskorpus angewendet und die dabei erzeugten Zielstrings, bzw. Fehlermeldungen (vgl. dazu Abschnitt 5.9) wurden im Anwendungskorpus neben den manuell vorgegebenen, korrekten Zielstrings abgespeichert. Durch Vergleich der maschinell erzeugten Zielstrings mit den vorgegebenen Zielstrings auf Übereinstimmung ermittelte ein Evaluierungsprogramm dann anschließend den prozentualen Anteil korrekt erzeugter Zielstrings an der Gesamtzahl der erzeugten Zielstrings, und es druckte diesen prozentualen Anteil aus. Eine von DERIVE anstelle eines Zielstrings generierte Fehlermeldung wurde dabei als nicht korrekt erzeugter Zielstring interpretiert.

Dieser Gesamtprozess wurde anschließend noch 9 mal wiederholt, und zwar jeweils ausgehend von einem Lernlauf zum 2., 3., ..., 9. und 10. Teil-Lernkorpus. Zur Veranschaulichung sei das Ganze mit einem Flußdiagramm wiedergegeben (vgl. Abb. 7.2):

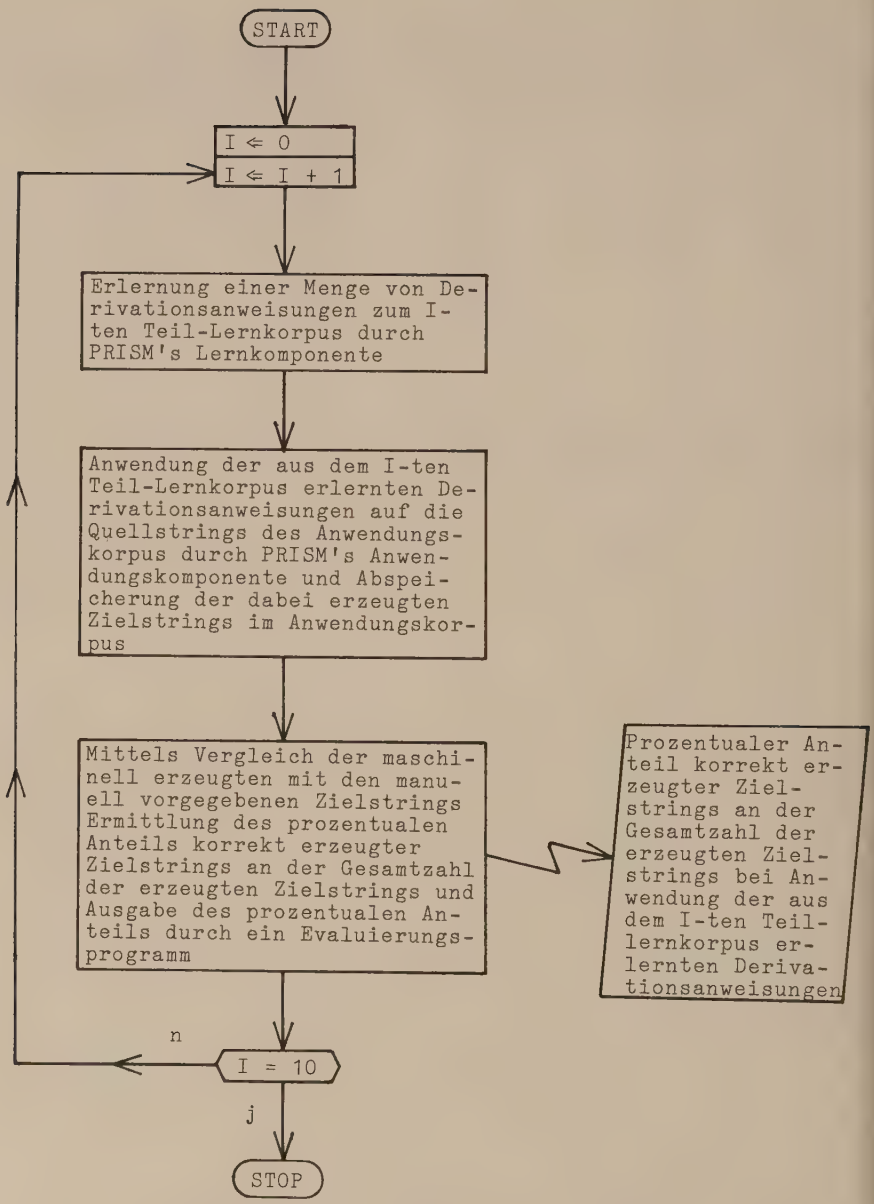


Abb. 7.2: Flußdiagramm zum Ablauf der Evaluierungsläufe zur 2. Forderung.

Mit den Evaluierungsläufen ergaben sich somit 10 Prozentwerte, die für jedes der 10 Teil-Lernkorpora angeben, wie groß der Anteil der *korrekten* Zielstrings an der Gesamtzahl der Zielstrings ist, die mit den aus diesen Teil-Lernkorpora erlernten Derivationsanweisungen zu den Quellstrings des Anwendungskorpus erzeugt wurden. Diese 10 Prozentwerte, die jeweils für die Wortpaarliste zur Motion und für die Verb-Nomen actionis-Liste ermittelt wurden, bilden die Grundlage für die Überprüfung der 2. Forderung.

Die für die Motion im Französischen ermittelten Werte geben die folgende Tabelle (Abb. 7.3) und - zur besseren Veranschaulichung - das sich anschließende Histogramm (Abb. 7.4) wieder. Dabei sind in der Tabelle für jedes Teil-Lernkorpus zusätzlich die Anzahl der erlernten Derivationsanweisungen und der Absolutwert korrekt erzeugter Zielstrings wiedergegeben.

Anzahl der Wortpaare im Teil-Lernkorpus	Prozentualer Anteil des Teil-Lernkorpus an der Lerngrundgesamtheit (n = 150)	Anzahl der erlernten Derivationsanweisungen	Korrekt erzeugte Zielstrings im Anwendungskorpus (n = 150)	
			proz.	abs.
15	10%	7	45,33%	68
30	20%	11	60,67%	91
45	30%	12	74,67%	112
60	40%	16	76,67%	115
75	50%	21	82,67%	124
90	60%	21	82,67%	124
105	70%	21	84,00%	126
120	80%	22	83,33%	125
135	90%	23	83,33%	125
150	100%	25	83,33%	125

Abb. 7.3: Statistische Ergebnisse der Evaluierungsläufe von PRISM zur 2. Forderung anhand der Motion im Französischen (Tabellarische Darstellung).

Abb. 7.4: Statistische Ergebnisse der Evaluierungsläufe von PRISM zur 2. Forderung anhand der Motion im Französischen (Histogramm).

Die statistischen Daten der Evaluierungsläufe mit der Ableitungsbeziehung zwischen Verben und Nomina actionis enthält folgende Tabelle (Abb. 7.5) und - wieder zur besseren Veranschaulichung - das umseitige Histogramm (Abb. 7.6):

Anzahl der Wortpaare im Teil-Lernkorpus	Prozentualer Anteil des Teil-Lernkorpus an der Lerngrundgesamtheit (n = 6.980)	Anzahl der erlernten Derivationsanweisungen	Korrekt erzeugte Zielstrings im Anwendungskorpus (n = 6.980)	
			proz.	abs.
698	10%	63	51,72%	3.610
1.396	20%	76	51,65%	3.605
2.094	30%	87	54,64%	3.814
2.792	40%	95	55,19%	3.852
3.490	50%	104	55,34%	3.863
4.188	60%	113	56,05%	3.912
4.886	70%	122	56,86%	3.969
5.584	80%	132	57,32%	4.001
6.282	90%	139	57,97%	4.046
6.980	100%	146	58,32%	4.071

Abb. 7.5: Statistische Ergebnisse der Evaluierungsläufe von PRISM zur 2. Forderung anhand der Ableitungsbeziehung zwischen Verben und Nomina actionis im Deutschen (Tabellarische Darstellung).

Die statistischen Ergebnisse in den Abbildungen 7.3 bis 7.6 dürften anschaulich zeigen, daß die 2. Forderung - zumindest für die beiden betrachteten Ableitungsbeziehungen - erfüllt ist, da im Mittel zu umso mehr an der jeweiligen Ableitungsbeziehung beteiligten und nicht im Lernkorpus enthaltenen Quellstrings der korrekte Zielstring erzeugt wurde, je größer das Lernkorpus wurde.

Ausgehend von den statistischen Ergebnissen läßt sich die Erfüllung der 2. Forderung allerdings nicht nur aufgrund der Anschauung, sondern auch mit einem exakten statistischen Maß überprüfen: Zu zeigen ist, daß der prozentuale Anteil (= y) korrekt erzeugter Zielstrings im Mittel um so größer wird, je größer der prozentuale Anteil (= x) des zugrunde liegenden Teil-Lernkorpus an der Lerngrundgesamtheit ist. Als statistisches Maß hierfür läßt sich der Regressionskoeffizient (= die Steigung) der Regressionsgeraden der y-Werte bezüglich der x-Werte verwenden. Wir berechnen den Regressionskoeffizienten b nach der

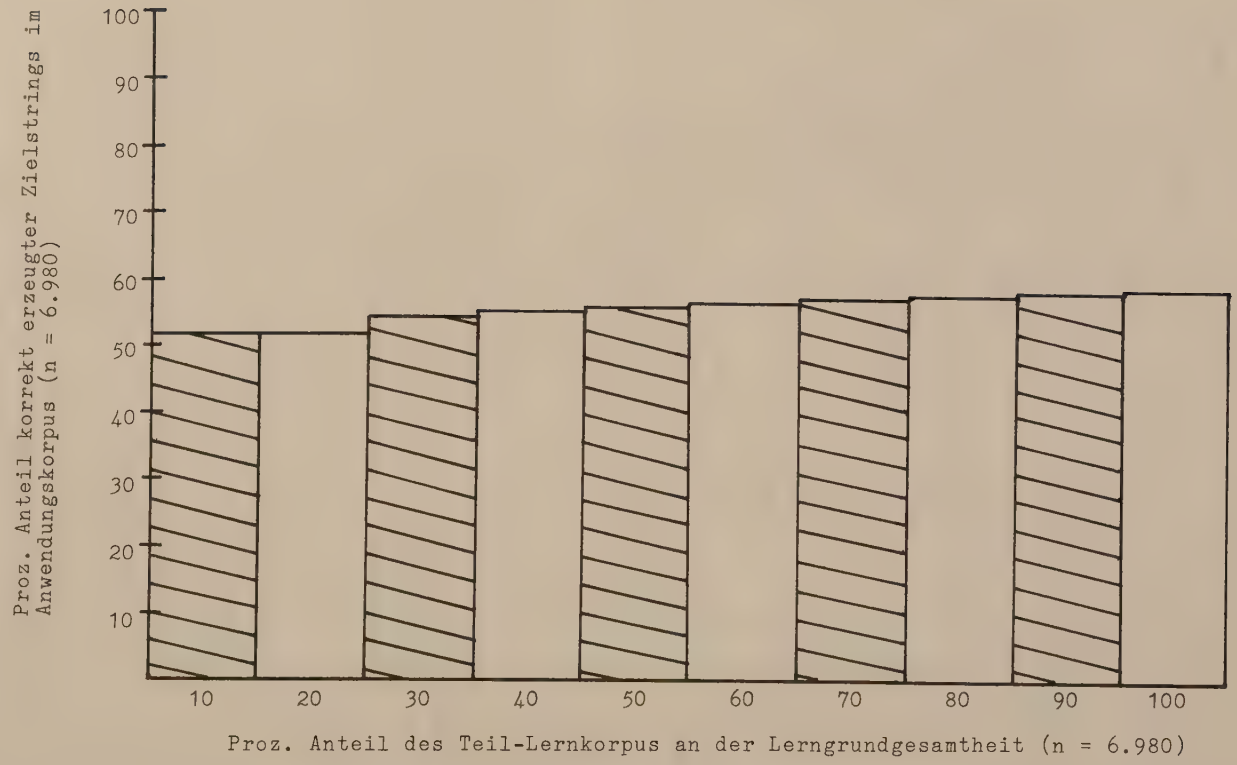


Abb. 7.6: Statistische Ergebnisse der Evaluierungsläufe von PRISM zur 2. Forderung anhand der Ableitungsbeziehung zwischen Verben und Nomina actionis im Deutschen (Histogramm).

Formel[1]

$$b = \frac{n \cdot \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2},$$

wobei
- n die Anzahl der (x,y)-Wertepaare ist,
- x_i der prozentuale Anteil des i-ten Teil-Lernkorpus an der Lerngrundgesamtheit ist,
- y_i der prozentuale Anteil derjenigen Zielstrings ist, die mit den aus dem i-ten Teil-Lernkorpus erlernten Derivationsanweisungen korrekt zu den Quellstrings des Anwendungskorpus erzeugt wurden.

Ist der Regressionskoeffizient
- größer als 0, so ist Forderung 2 bestätigt.
- kleiner oder gleich 0, so ist Forderung 2 nicht erfüllt.

Bei Einschluß des Ursprungs (0%, 0%) als 11. Wertepaar ergibt sich für die Motion der Regressionskoeffizient 0,60. Für die Ableitungsbeziehung zwischen Verben und Nomina actionis erhalten wir 0,31. Forderung 2 ist somit erfüllt.

An den Statistiken in den Abbildungen 7.3 bis 7.6 läßt sich weiterhin ablesen, daß PRISM die in 6.1 genannte Erwartung, die wir mit der Wahl zweier unterschiedlich regelhafter Ableitungsbeziehungen verbunden hatten, erfüllt: Für die auf graphischer Ebene wesentlich regelhaftere Motion liefert es bessere Ergebnisse als für die weniger regelhafte Ableitungsbeziehung zwischen Verben und Nomina actionis. Bei der Motion wird im Mittel zu 75,67% der nicht im Lernkorpus enthaltenen Quellstrings der korrekte Zielstring gebildet, bei der Ableitungsbeziehung zwischen Verben und Nomina actionis dagegen im Mittel nur zu 55,51%.

Die qualitative Auswertung der fehlerhaft erzeugten Zielstrings ergab sowohl bei der Motion als auch bei der Ableitungsbeziehung zwischen Verben und Nomina actionis, daß beide theoretisch denkbaren Typen von Fehlern vertreten sind:

1 Nach Kreyszig 1979: 261.

1. *Zu Quellstrings, die selteneren Bildungsmustern angehören, wurden fälschlich nach regelhafteren Bildungsmustern Zielstrings erzeugt.*
   So wurde beispielsweise bei den Evaluierungsläufen zur Motion zu 'âne' <Esel> anstatt der korrekten weiblichen Form 'ânesse' die inkorrekte Form *'âne' gebildet, und zwar nach einem der häufigsten Muster zur Bildung weiblicher Formen (i. allg. ist die weibliche Form gleich der männlichen Form, falls die männliche Form auf 'e' endet). Oder es wurde zu 'fils' <Sohn> anstatt der korrekten, nach einem Ausnahmemuster gebildeten weiblichen Form 'fille' inkorrekt die Form *'filse' nach dem häufigsten Bildungsmuster erzeugt ("Das *normale Kennzeichen* für das Femininum ist die Endung 'e'." (Klein/Strohmeyer 1979: 99)).

   Bei der Ableitungsbeziehung zwischen Verben und Nomina actionis trat dieser Fehlertyp u. a. oft bei impliziten Ableitungen auf. Anstelle der korrekten Bildung eines Nomen actionis durch implizite Ableitung wurde dann z. B. ein Nomen actionis mittels der häufigeren Suffigierung mit 'ung' gebildet. So wurde etwa zu 'abbauen' anstatt der korrekten Form 'Abbau' die Form *'Abbauung' erzeugt.

2. *Zu Quellstrings, die regelhafteren Bildungsmustern angehören, wurden fälschlich nach selteneren Bildungsmustern Zielstrings erzeugt.*
   Bei der Motion ergab sich dieser Fehler z. B. für 'avocat' <Rechtsanwalt>. Die korrekte weibliche Form 'avocate' wird nach dem häufigsten Muster durch Anhängen von 'e' gebildet. Statt dieser Form wurde *'avocatte' erzeugt, und zwar nach einem Bildungsmuster, an dem nach unseren Recherchen nur ein Wortpaar beteiligt ist, nämlich ('chat' <Kater>, 'chatte').

   Ein inkorrektes Nomen actionis wurde zu 'abbüßen' durch Anwendung eines selteneren Bildungsmusters generiert. Korrekte Form wäre 'Abbüßung'. Statt dessen wurde nach einem selteneren Muster - in Analogie zu 'büßen' <==> 'Buße' - die nicht gebräuchliche Form *'Abbuße' gebildet.

Fehlertyp 1 trat bedeutend häufiger auf als Fehlertyp 2, und

zwar bei der Motion ca. 6 mal so häufig, bei der Nominalisierung von Verben ca. 5 mal so häufig. Eine plausible Erklärung für diesen Häufigkeitsunterschied scheint zu sein, daß bei linguistisch *zufällig* erstellten Lernkorpora - wie sie bei den Evaluierungsläufen zur 2. Forderung verwendet wurden - im Mittel eher die selteneren Bildungsmuster der Sprache nicht im Lernkorpus vertreten sind als die regelhafteren, also häufigeren Bildungsmuster der Sprache, so daß leichter für die selteneren Bildungsmuster der Sprache keine entsprechenden Derivationsanweisungen erlernt werden können als für die regelhafteren und es zu einer Übergeneralisierung der Derivationsanweisungen zu den regelhafteren Bildungsmustern kommt. Treten dann bei der späteren Anwendung der Anweisungen im Anwendungskorpus Quellstrings auf, die den selteneren Mustern angehören, so fehlen die entsprechenden Derivationsanweisungen und es kommen fälschlich die Anweisungen für die regelhafteren Muster zum Zuge.

Entsprechende Fehler durch Übergeneralisierung regelhafter Bildungsmuster treten auch beim menschlichen Erstspracherwerb auf, und man kann hier vom Rechner wohl kaum mehr erwarten als vom menschlichen Lerner. Daß wir beim Spracherwerb tatsächlich übergeneralisierende derivations- und flexionsmorphologische Regeln konstruieren, aus denen inkorrekte Wortbildungen und Flexionsformen resultieren, dürfte jeder wissen, der schon einmal den Erstspracherwerb eines Kindes mitverfolgt hat. Fehlerhafte Bildungen infolge übergeneralisierender Übertragung regelhafter Bildungsmuster auf Wörter, die unregelmäßigen Mustern angehören, sind dabei keine Seltenheit. Dies belegen auch wissenschaftliche Untersuchungen von Jean Berko (1958), Bruce L. Derwing (1976) und Bruce L. Derwing/William J. Baker (1977 u. 1979).

Beispielsweise nennt Derwing (1976: 40) für das Englische die belegte inkorrekte Ableitung der Nomina agentis

*'cooker' und *'teaser'

aus den Verben 'cook' <kochen> und 'tease' <necken>. Richtig wären hier die unregelmäßig gebildeten Nomina agentis 'cook' <Koch> und 'tease' <Necker>. Das dominierende Bildungsmuster

für Nomina agentis besteht im Anhängen des Vokals /ə/ an den Verbstamm. In der Orthographie drückt sich dies

- durch Anhängen von 'er' an nicht auf 'e' endende Verbstämme aus (wie z. B. bei der Ableitung von 'teacher' <Lehrer> aus 'teach' <lehren>)
- sowie durch Anhängen von 'r' an Verbstämme, die auf 'e' enden (z. B. bei der Ableitung von 'driver' <Fahrer> aus 'drive' <fahren>).

Ausgehend von diesem Bildungsmuster hat der Lerner, der obige inkorrekte Nomina agentis bildete, offensichtlich eine übergeneralisierende *derivations*morphologische Regel gebildet und auf Verben angewandt, deren entsprechende Nomina agentis unregelmäßig gebildet werden.

Die Existenz übergeneralisierender *flexions*morphologischer Regeln belegen Untersuchungsergebnisse von Jean Berko (1958: 175). Sie konnte z. B. feststellen, daß manche Kinder im Englischen zum Verbum 'ring' <klingeln> anstelle des korrekten stark flektierten Präteritums 'rang' <klingelte> die nicht korrekte Form *'ringed' bilden, und zwar in Analogie zu dem bei den englischen Verben häufiger vertretenen schwach flektierten Präteritum.

3. Forderung

Um die 3. Forderung überprüfen zu können, wurde zunächst sowohl für die Motion im Französischen als auch für die Ableitungsbeziehung zwischen Verben und Nomina actionis im Deutschen ein linguistisch annähernd repräsentatives[2] Lernkorpus erstellt. Strenggenommen hätte dabei jeweils ein Lernkorpus erstellt werden müssen, das für die betreffende Grundgesamtheit repräsentativ ist, also für die Gesamtheit der an der betreffenden Ableitungsbeziehung in der Sprache beteiligten Wortpaare. Aus dem Lernkorpus hätte dann eine Menge von Derivationsanweisungen erlernt werden müssen und anschließend auf alle nicht im Lernkorpus enthaltenen Quellstrings, die in der Sprache an der Ableitungsbeziehung beteiligt sind, angewendet werden müssen. Wegen der hohen Anzahl

2 Zum hier verwendeten Repräsentativitätsbegriff vgl. Abschnitt 3.2.

der an den beiden betrachteten Ableitungsbeziehungen beteiligten Wortpaare, wäre dies ein Unterfangen gewesen, das den zeitlichen Rahmen der vorliegenden Arbeit gesprengt hätte. Wir haben uns daher darauf beschränkt, 2 Lernkorpora zu erstellen, die lediglich für unsere 2 Wortpaarlisten zur Motion und zur Ableitungsbeziehung zwischen Verben und Nomina actionis repräsentativ sind.

Das für die Wortpaarliste zur Motion repräsentative Lernkorpus wurde folgendermaßen gewonnen:

1. Mittels einer Variante von PRISM wurde ein Lernlauf mit der Wortpaarliste durchgeführt. Dabei wurden die Quellstrings all derjenigen Wortpaare,
   - die in DISCOV die Konstruktion des Substitutionsmusters einer Substitutionsanweisung auslösten
   - oder die die Konstruktion einer Kontextbedingung auslösten,

   in eine Datei geschrieben. Mittels dieser Datei könnte man aus der Wortpaarliste zur Motion ein Lernkorpus extrahieren, das zwar für die implizit (in der Liste) auftretenden Substitutionsmuster repräsentativ ist, jedoch nicht für die Impedienten und die Häufigkeitsrangfolge der Substitutionsmuster.
2. In einer intellektuellen Postedition wurde die Datei daher so um Quellstrings erweitert,
   - daß auch alle in der Wortpaarliste implizit vertretenen Typen von Impedienten in der Datei implizit auftreten
   - und daß in dem mit der Datei extrahierbaren Lernkorpus die Häufigkeitsrangfolge zumindest derjenigen Substitutionsmuster mit ihrer Häufigkeitsrangfolge in der Wortpaarliste übereinstimmt, deren Substituenda linke (bei präfixorientierten Substitutionsanweisungen) oder rechte (bei suffixorientierten Substitutionsanweisungen) Teilstrings voneinander sind.
3. Mit Hilfe der so erstellten Quellstringliste wurde die Wortpaarliste zur Motion in ein (für diese Wortpaarliste) repräsentatives Lernkorpus und in ein Anwendungskorpus getrennt:

- Alle diejenigen Wortpaare der Wortpaarliste, deren Quellstrings in der Quellstringliste enthalten waren, bildeten das *repräsentative Lernkorpus*.
- Alle diejenigen Wortpaare der Wortpaarliste, deren Quellstrings *nicht* in der Quellstringliste enthalten waren, bildeten das *Anwendungskorpus*.

Zur Verb-Nomen actionis-Liste wurde das repräsentative Lernkorpus auf folgende Weise gewonnen:

1. Zunächst wurde wieder mit einer Variante von PRISM ein Lernlauf mit der Liste durchgeführt, und es wurden die Quellstrings all derjenigen Wortpaare in eine Datei geschrieben,
   - die in DISCOV die Konstruktion des Substitutionsmusters einer Substitutionsanweisung auslösten
   - oder die die Konstruktion einer Kontextbedingung auslösten.

   Mittels dieser Datei ließe sich aus der Verb-Nomen actionis-Liste ein Lernkorpus extrahieren, das zwar für die in der Liste implizit auftretenden Substitutionsmuster und Kontextbedingungen repräsentativ ist, jedoch nicht für die Häufigkeitsrangfolge der Substitutionsmuster.
2. In einer halbautomatischen Postedition wurde die Datei daher so um Quellstrings erweitert, daß in dem mit der Datei extrahierbaren Lernkorpus die Häufigkeitsrangfolge zumindest der häufigsten derjenigen Substitutionsmuster mit ihrer Häufigkeitsrangfolge in der Verb-Nomen actionis-Liste übereinstimmt, deren Substituenda linke (bei präfixorientierten Substitutionsanweisungen) oder rechte (bei suffixorientierten Substitutionsanweisungen) Teilstrings voneinander sind.
3. Mit Hilfe der so erstellten Quellstringliste wurde - analog zum Vorgehen bei der Wortpaarliste zur Motion (s. o.) - die Verb-Nomen actionis-Liste in ein repräsentatives Lernkorpus und in ein Anwendungskorpus getrennt.

Da in der Verb-Nomen actionis-Liste keine Wortpaare mit leerem Zielstring auftreten, enthält sie (implizit) auch keine Impedienten. Auf implizit in der Liste auftretende Impedienten mußte daher bei der Erstellung des repräsentativen Lernkorpus nicht geachtet

werden.

Ausgehend von den so gewonnenen Lern- und Anwendungskorpora wurden die Evaluierungsläufe zur 3. Forderung für die Motion im Französischen und für die Ableitungsbeziehung zwischen Verben und Nomina actionis im Deutschen durchgeführt. Die für die Überprüfung der 3. Forderung relevanten statistischen Daten gibt folgende Tabelle wieder:

	Wortpaarliste zur Motion	Verb-Nomen actionis-Liste
Anzahl der Wortpaare im repräsentativen Lernkorpus	89	9.167
Anzahl der Derivationsanweisungen, die vom Lernalgorithmus aus dem Lernkorpus gewonnen wurden	35	198
Anzahl der Quellstrings im Anwendungskorpus	211	4.793
Prozentualer Anteil derjenigen Zielstrings, die mittels der erlernten Derivationsanweisungen korrekt zu den Quellstrings des Anwendungskorpus generiert wurden, an der Gesamtzahl der zu den Quellstrings des Anwendungskorpus generierten Zielstrings	100%	100%

Abb. 7.7

Die in der Tabelle wiedergegebenen Prozentwerte für den Anteil korrekt erzeugter Zielstrings verdeutlichen, daß auch die 3. Forderung - zumindest für die beiden betrachteten Ableitungsbeziehungen - erfüllt ist: Die aus den repräsentativen Lernkorpora gewonnenen Derivationsanweisungen erzeugen zu 100% der an den betreffenden Ableitungsbeziehungen beteiligten und nicht in den Lernkorpora enthaltenen Quellstrings unserer Wortpaarlisten die korrekten Zielstrings. Die Mengen von Derivationsanweisungen

sind also stark adäquat.

Sicher fragt man sich, warum das repräsentative Lernkorpus für die Verb-Nomen actionis-Liste wesentlich umfangreicher ist als das für die Wortpaarliste zur Motion. Der Grund dafür ist, daß die Ableitungsbeziehung zwischen Verben und Nomina actionis i. U. zur Motion recht wenige Regelmäßigkeiten auf graphischer Ebene aufweist, d. h., an ihr sind wesentlich mehr verschiedene Substitutionsmuster beteiligt, die jeweils für verhältnismäßig wenige Verben gelten und die auch so gut wie nicht graphisch oder phonologisch konditioniert sind, so daß zu Substitutionsmustern, deren Substituenda Teilstrings voneinander sind, i. allg. überaus komplexe Kontextbedingungen aufgestellt werden müssen. Per definitionem müssen in einem *repräsentativen* Lernkorpus all diese Substitutionsmuster und Kontextbedingungen implizit vertreten sein. Entsprechend wurde das für die Ableitungsbeziehung zwischen Verben und Nomina actionis repräsentative Lernkorpus bedeutend umfangreicher als das für die Motion.

4. Forderung

Ebenso wie die 2. und die 3. Forderung hätte diese Forderung strenggenommen für alle morphologischen Ableitungsbeziehungen in allen natürlichen Sprachen überprüft werden müssen. Da dies praktisch nicht durchführbar ist, mußten wir uns auch hier auf eine exemplarische Überprüfung beschränken. Wir haben die Forderung anhand der Motion im Französischen überprüft und anhand der Negation englischer Adjektive mit dem Präfix 'in' und seinen Varianten. Die Überprüfung anhand der Ableitungsbeziehung zwischen Verben und Nomina actionis kam nicht in Frage, da aufgrund ihrer geringen Regelhaftigkeit keine vollständige linguistische Beschreibung dieser Ableitungsbeziehung vorliegt, mit der die von PRISM entdeckten Regeln hätten verglichen werden können, um zu überprüfen, ob der Lernalgorithmus auch diejenigen Bildungsmuster als regelhaft klassifiziert, die Linguisten als regelhaft klassifizieren.

Zunächst zur Motion im Französischen: Zur Überprüfung der 4. Forderung wurde PRISM's Lernkomponente auf die vorliegende Wortpaarliste zur Motion angewendet und die erlernten Derivations-

anweisungen wurden mit der Angabe ihrer Anwendungshäufigkeit ausgedruckt, d. h., zu jeder ermittelten Derivationsanweisung wurde mit ausgedruckt, auf wieviele Quellstrings der Wortpaarliste sie anwendbar ist. Das eingegebene Lernkorpus und das Anwendungsergebnis der Lernkomponente sind im Anhang (10.2) abgedruckt. Wir werden uns im folgenden auf 'PAGE 4' des Ergebnisausdruckes beziehen. Als Bildungsmuster, die der Lernalgorithmus als die regelhafteren klassifiziert hat, sind diejenigen anzusehen, die durch Derivationsanweisungen mit hohen Anwendungshäufigkeiten beschrieben sind. Es handelt sich dabei um die präfixorientierte Substitutionsanweisung

1. '' → ''/#__ (96,33%)

und um die suffixorientierten Substitutionsanweisungen

2. 'x' → 'se'/__# (1,67%)
3. 'ur' → 'resse'/('ande'|'fende'|'enge'|'<che'|'asse')__# (1,67%)
4. '' → 'sse'/('ce'|'/ne'|'gre'|'tre'|'/te')__# (3,00%)
5. '' → 'ne'/('ysan'|'en'|'on')__# (6,67%)
6. '' → ''/'e'__# (10,33%)
7. 'r' → '>re'/'e'__# (11,67%)
8. 'eur' → 'rice'/('ad'|'at'|'ct'|'it'|'ent'|'cont'|'ot'|'empt'|'upt'|'cut')__# (17,33%)
9. 'r' → 'se'/'u'__# (19,00%)
10. '' → 'e'/__# (19,67%)

Rechts von den Anweisungen ist als Prozentwert die jeweilige Anwendungshäufigkeit angegeben. Die in den Anweisungen innerhalb von objektsprachlichen Zeichenketten auftretenden Zeichen '<', '>', '/' kodieren respektive die Akzentzeichen '´', '`', '^'. Sie beziehen sich auf einen unmittelbar links von ihnen stehenden Vokalbuchstaben, der jedoch in den Anweisungen nicht auftreten muß, wohl aber in den Quell- und/oder Zielstrings, aus denen die Anweisungen gewonnen wurden, bzw. in den Quellstrings, auf die sie angewendet werden.

Zur Überprüfung von Forderung 4 wurden die 10 Anweisungen mit den Bildungsmustern verglichen, die in der "Französischen

Sprachlehre" von H. W. Klein und F. Strohmeyer (1979: 99ff.) als regelhaft angegeben sind. Außer der präfixorientierten Substitutionsanweisung (1) und der ersten suffixorientierten Substitutionsanweisung (2) sind alle und nur die durch diese Anweisungen beschriebenen Bildungsmuster auch von Klein/Strohmeyer als regelhaft beschrieben. Eine Entsprechung zu Anweisung 1 wurde von Klein/Strohmeyer möglicherweise als überflüssig angesehen, da diese Anweisung lediglich beschreibt, daß sich bei der Motion i. allg. nichts am Wortanfang ändert. Anweisung 2, die z. B. für 'époux' <Gatte> <==> 'épouse' gilt, scheint von Klein/Strohmeyer übersehen worden zu sein. Eine entsprechende Regel findet sich nämlich in der Grammatik "Le bon usage" von Maurice Grevisse (1964: 173f.).

Die häufigste Methode zur Bildung der weiblichen Form besteht nach Klein/Strohmeyer im Anhängen von 'e' an die männliche Form ("Das *normale Kennzeichen* für das Femininum ist die Endung 'e'." (Klein/Strohmeyer 1979: 99); Beispiel: 'ami' <Freund> <==> 'amie'). Dem entspricht auch diejenige Substitutionsanweisung mit der höchsten Anwendungshäufigkeit (10). Daneben führen sie weitere regelhafte Bildungsmuster ohne Angabe ihrer Anwendungshäufigkeit an:

- "'*eur*' - '*eresse*'" (Klein/Strohmeyer 1979: 101; Beispiel: 'chasseur' <Jäger> <==> 'chasseresse'). Diese Beschreibung besagt, daß zu männlichen Substantiven, welche auf 'eur' enden, die weibliche Form mittels Ersetzung dieser Endung durch 'eresse' gebildet wird. (Analoges gilt für die folgenden in der gleichen Notation wiedergegebenen Regeln.) Dieser Regel entspricht die Substitutionsanweisung 3. Allerdings liefert Anweisung 3 zum einen eine adäquatere, zum anderen eine redundanzfreiere Beschreibung:

  Adäquater ist sie, da sie aufgrund ihrer Kontextbedingung i. U. zur Beschreibung von Klein/Strohmeyer beispielsweise die Bildung der inkorrekten weiblichen Form *'promeneresse' anstelle von 'promeneuse' zu 'promeneur' <Spaziergänger> nicht erlaubt.

  Weiterhin gibt Anweisung 3 eine *redundanzfreiere* Beschreibung als Klein/Strohmeyer: Im Unterschied zu ihrer Beschrei-

bung muß nämlich strenggenommen nicht 'eur' durch 'eresse' ersetzt werden, sondern - wie in Anweisung 3 angegeben - nur 'ur' durch 'resse', da ja die Ersetzung von 'e' durch 'e' nichts verändert.

Ähnliche Inadäquatheiten und Redundanzen weisen auch einige der im folgenden wiedergegebenen Beschreibungen von Klein/Strohmeyer auf. Wir wollen diese Unzulänglichkeiten allerdings nicht weiter verfolgen und belassen es bei ihrer exemplarischen Illustration am Muster 'eur' - 'eresse'. Im übrigen scheinen diese beiden Unzulänglichkeiten bei der eher didaktischen Zielsetzung, die Klein/Strohmeyer mit ihrer für den französischen Sprachunterricht konzipierten Grammatik verfolgen, auch tolerierbar und zum Teil notwendig zu sein.

- "'*e*' - '*esse*'" (Klein/Strohmeyer 1979: 101; Beispiel: 'prince' <Prinz> <==> 'princesse'). Dieser Regel entspricht Anweisung 4.
- "'*an*' - '*anne*', '*on*' - '*onne*', '*ien*' - '*ienne*'" (Klein/Strohmeyer 1979: 100; Beispiele: 'paysan' <Bauer> <==> 'paysanne', 'baron' <Baron> <==> 'baronne', 'musicien' <Musiker> <==> 'musicienne'). Diese Regel wird durch Anweisung 5 repräsentiert.
- "Die meisten *Substantive, die auf 'e' enden,* haben in Maskulinum und Femininum gleiche Form." (Klein/Strohmeyer 1979: 99; Beispiel: 'camarade' <Gefährte> <==> 'camarade'). Eine Beschreibung dieses Bildungsmusters liefert Anweisung 6.
- "'*er*' - '*ère*'" (Klein/Strohmeyer 1979: 100; Beispiel: 'berger' <Schäfer> <==> 'bergère'). Dieser Regel entspricht Anweisung 7.
- "'*teur*' - '*trice*'" (Klein/Strohmeyer 1979: 101; Beispiel: 'acteur' <Schauspieler> <==> 'actrice'). Diese Regel gibt Anweisung 8 wieder.
- "'*eur*' - '*euse*'" (Klein/Strohmeyer 1979: 100; Beispiel: 'danseur' <Tänzer> <==> 'danseuse'). Eine Beschreibung dieses Bildungsmusters gibt Anweisung 9.

Der Vergleich der maschinell generierten Derivationsanweisungen mit der Beschreibung von Klein/Strohmeyer zeigt also, daß

Forderung 4 für die Motion im Französischen erfüllt ist. - Um falschen Vorstellungen vorzubeugen, sei hier nachdrücklich darauf hingewiesen, daß ein Lernalgorithmus keineswegs *selbstverständlich oder gar logisch notwendig* in einem repräsentativen Lernkorpus diejenigen Bildungsmuster als die häufigeren und somit regelhafteren identifiziert, die auch Linguisten als regelhaft klassifizieren. Wenn PRISM etwa die unerwünschte Eigenschaft hätte, nicht die kürzestmöglichen, sondern längere Substituenda und Substituenten zu ermitteln, so würde es eine andere Häufigkeitsverteilung der Bildungsmuster feststellen, als sie aus linguistischer Sicht im Lernkorpus gegeben ist, und entsprechend würden andere Bildungsmuster als regelhafter klassifiziert.

Bei der zweiten Ableitungsbeziehung, die wir zur Überprüfung der 4. Forderung herangezogen haben, handelt es sich um die Negation englischer Adjektive mit dem Präfix 'in' und seinen Varianten 'im', 'il' und 'ir'. Das eingegebene Lernkorpus für diese Ableitungsbeziehung und das Anwendungsergebnis findet man im Anhang (10.3). Die erlernten Derivationsanweisungen sind auf 'PAGE 4' des Ergebnisausdruckes wiedergegeben. Der Vergleich dieser Anweisungen mit der entsprechenden Beschreibung in der englischen Wortbildungslehre "The categories and types of present-day English word-formation" von Hans Marchand (1969) zeigt, daß Forderung 4 auch für die Negation englischer Adjektive mittels 'in' und seinen Varianten erfüllt ist. Marchand (1969: 169) schreibt:

> "At all times 'in' has been prefixed in accordance with Latin rules, i. e. 'in' is assimilated to 'il' or 'i' before 'l', to 'im' before 'm', 'b', 'p', to 'ir' or 'i' before 'r': 'illegible', 'illimitable', 'illoricated', 'immarginate', 'impennate', 'imponderable', 'irrebuttable', 'irreciprocal', 'irreplaceable', etc."

Diesen Regeln entsprechen im wesentlichen auch die erlernten präfixorientierten Substitutionsanweisungen:

```
'' → 'il'/# __'l'
'' → 'ir'/# __'r'
'' → 'im'/# __ ('m'|'p')
'' → 'in'/# __
```

Zu Marchands Beschreibung weisen diese Anweisungen 3 - hier irrelevante - Abweichungen auf. Sie beschreiben nicht:

- die Assimilation von 'in' zu 'i' vor 'l',
- die Assimilation von 'in' zu 'im' vor 'b',
- die Assimilation von 'in' zu 'i' vor 'r'.

Dies ist darauf zurückzuführen, daß Marchands Beschreibung nicht nur für Adjektive, sondern auch für Verben und Substantive gilt. Die Derivationsanweisungen beschreiben jedoch nur die Negation der *Adjektive*, bei denen diese 3 Assimilationen nicht vorkommen.

5. Forderung

Die 5. Forderung, die Redundanzfreiheit der erlernten Mengen von Derivationsanweisungen verlangt, konnte als einzige Forderung nicht erfüllt werden. Zwar ergab sich bei der Überprüfung von Forderung 4 anhand der Motion im Französischen, daß PRISM durchaus redundanzfreiere Beschreibungen liefern kann, als sie in einer Grammatik auftreten. Auch zeigte die Beschreibung von DISCOV in Abschnitt 5.4, daß die verwendeten Lernstrategien zumindest die Tendenz haben, möglichst kurze Mengen von Derivationsanweisungen zu liefern. An einem Beispiel läßt sich jedoch leicht zeigen, daß PRISM nicht immer die kürzeste Beschreibung liefert, die in der vorgegebenen Beschreibungssprache möglich ist. Nehmen wir an, dem Lernalgorithmus würde das folgende - linguistisch keineswegs repräsentative - Lernkorpus zur Motion im Französischen vorgegeben:

```
('acteur'     <Schauspieler>, 'actrice'),
('conducteur' <Fahrer>,       'conductrice'),
('spectateur' <Zuschauer>,    'spectatrice'),
('chasseur'   <Jäger>,        'chasseresse'),
('pécheur'    <Sünder>,       'pécheresse').
```

Zu diesem Korpus würde die Lernkomponente von PRISM neben der präfixorientierten Substitutionsanweisung '' → ''/#__ die 2 folgenden suffixorientierten Substitutionsanweisungen erlernen:

```
'ur'  → 'resse'/('he'|'se') __#
'eur' → 'rice'/__#
```

Da das Substituendum des Substitutionsmusters 'ur' → 'resse' rechter Teilstring des Substituendums 'eur' des zweiten Substitutionsmusters ist und da das Muster 'ur' → 'resse' seltener

als 'eur' → 'rice' im Lernkorpus vertreten ist, wird zu 'ur' → 'resse' eine Kontextbedingung erstellt. Mit 'eur' → 'rice' bildet DISCOV dagegen eine kontextfreie Substitutionsanweisung. Die resultierende Menge suffixorientierter Substitutionsanweisungen ist nicht redundanzfrei. Würde nämlich zum Substitutionsmuster 'eur' → 'rice' eine kontextsensitive Substitutionsanweisung konstruiert und zu 'ur' → 'resse' eine kontextfreie Anweisung, so ergäbe sich die kürzere und auch schwach adäquate Beschreibung:

'eur' → 'rice'/'t'__#
'ur' → 'resse'/__#

Nun stellt sich bestimmt die Frage, warum DISCOV nicht so konstruiert wurde, daß es die zweite, redundanzfreie Menge von Substitutionsanweisungen erlernt. Neben der Forderung nach Redundanzfreiheit hatten wir noch 2 weitere Forderungen (2 u. 3) aufgestellt, die verlangen, daß die erlernten Mengen von Derivationsanweisungen zu möglichst vielen Quellstrings, die an der jeweils beschriebenen morphologischen Ableitungsbeziehung beteiligt, jedoch nicht im Lernkorpus enthalten sind, korrekte Zielstrings erzeugen. Um dies zu erreichen, haben wir die Lernstrategien in DISCOV so angelegt, daß sie zu den häufiger im Lernkorpus vertretenen Substitutionsmustern (, von denen immer angenommen wird, daß sie auch in der Sprache häufiger auftreten) allgemeinere, also weniger kontextsensitive Substitutionsanweisungen konstruieren als zu den selteneren Substitutionsmustern. Hierdurch wird - bei Repräsentativität des Lernkorpus - erreicht, daß die Substitutionsanweisungen, die zu den häufigeren Mustern konstruiert wurden, auf mehr nicht im Lernkorpus enthaltene Quellstrings anwendbar sind als diejenigen zu den selteneren Mustern, wodurch wiederum die Wahrscheinlichkeit wächst, daß mehr korrekte Zielstrings erzeugt werden. Diese zur Erfüllung der Forderungen 2 und 3 erforderliche Strategie hat im Beispiel zu einer redundanten Menge suffixorientierter Substitutionsanweisungen geführt, die Forderung 5 nicht erfüllt. Würde PRISM die zweite, redundanz*freie* Menge suffixorientierter Substitutionsanweisungen erlernen, so wäre zwar Forderung 5 erfüllt, je-

doch würde die Erfüllbarkeit der Forderungen 2 und 3 bedeutend unwahrscheinlicher. Das obige Beispiel zeigt also, daß Forderung 5 nicht für alle Lernkorpora gleichzeitig mit den Forderungen 2 und 3 erfüllt werden kann.

8 ANWENDUNGSMÖGLICHKEITEN VON PRISM

Wie in Abschnitt 1.1 ausgeführt handelt es sich bei dieser Arbeit um *Grundlagenforschung* für einen anwendungsorientierten Algorithmus. Entsprechend soll PRISM noch kein anwendungsorientierter Algorithmus sein. Vielmehr soll es als experimentelles System

- die Entwicklung einiger wesentlicher Lernstrategien ermöglichen, die für einen anwendungsorientierten Algorithmus erforderlich sind,
- und es soll noch offene Probleme sichtbar machen (vgl. dazu Abschnitt 9).

Bei den im folgenden genannten Anwendungsmöglichkeiten handelt es sich daher weniger um Anwendungsmöglichkeiten von PRISM, als vielmehr um solche des anvisierten anwendungsorientierten Algorithmus, für den PRISM die wesentlichen Grundlagen liefern soll. Die *primär* intendierten Anwendungsmöglichkeiten wurden bereits in Abschnitt 1.1 genannt. Sie seien deshalb hier nur in Stichworten wiederholt:

1. Maschinelle Erstellung von Algorithmen für die automatische Generierung abgeleiteter Wortformen.
2. Bereitstellung von Daten für die Erstellung einer Grammatik der Derivationsmorphologie einer Sprache.
3. Messung der Regelhaftigkeit morphologischer Ableitungsbeziehungen auf graphischer Ebene.

Die erste Anwendung dürfte z. B. im Rahmen linguistisch orientierter Informationssysteme von Interesse sein. Wie die Darstellung der Arbeiten von Pacak/Pratt und in den Projekten ISLIB und JUDO (vgl. Abschnitt 2.1.3 - 2.1.5) gezeigt hat, erhofft man sich bei diesen Systemen von der maschinellen Simulation morphologischer Ableitungsregeln eine Erhöhung des Recall beim Retrieval. Die erforderlichen Algorithmen zur Simulation der Ableitungsregeln könnten mit dem anvisierten Lernalgorithmus bedeutend

leichter *maschinell* erstellt werden, als es die herkömmliche, rein *intellektuelle* Erstellung der Algorithmen erlaubt.

Die Anwendungen 2 und 3 könnten bei sprachwissenschaftlichen Untersuchungen von Nutzen sein.

Neben diesen zunächst intendierten Anwendungsmöglichkeiten haben sich bei der experimentellen Arbeit mit PRISM folgende weitere Anwendungsmöglichkeiten für den anwendungsorientierten Algorithmus ergeben:

4. Er könnte zur Erlernung und Simulation von Deflexionsregeln zur automatischen Lemmatisierung verwendet werden. Eingabe der Lernkomponente wären dann nicht Wortpaare, zwischen deren Komponenten eine morphologische Ableitungsbeziehung besteht, sondern Wortpaare, deren Erstkomponente eine flektierte Wortform ist und deren Zweitkomponente die dazugehörige Grundform ist. Ausgehend von solch einem Lernkorpus könnte die Lernkomponente dann eine Menge von Anweisungen erlernen, die die Erzeugung der Grundformen aus den Flexionsformen beschreibt. Mit diesen Anweisungen könnte die Anwendungskomponente des Systems bezugnehmend auf die graphische Gestalt der Flexionsformen eine wortformenbezogene Lemmatisierung durchführen. Diese Anwendung kommt allerdings nur in Frage, wenn man bereit ist, die Nachteile einer wortformenbezogenen maschinellen Lemmatisierung in Kauf zu nehmen und ggf. durch eine Postedition auszugleichen. Nachteile einer solchen wortformenbezogenen Lemmatisierung sind z. B., daß keine automatische Homographentrennung durchgeführt werden kann und daß diskontinuierliche Konstituenten nicht zusammengeführt werden können, da für diese beiden Prozesse auch eine syntaktische und ggf. auch semantische Analyse auf Satzebene erforderlich ist. - Daß eine wortformenbezogene Lemmatisierung, *die vornehmlich auf die graphische Gestalt der Flexionsformen Bezug nimmt*, trotz ihrer linguistischen Mängel auch tatsächlich nutzbringend angewendet werden kann, zeigen die Anwendungen des Programmsystems LEMMA, das Gerd Willée für die Lemmatisierung gegenwartsdeutscher Texte entwickelt hat (vgl. dazu Krause/Willée 1981).

5. Der Algorithmus könnte zur Erlernung und Simulation von Regeln eingesetzt werden, die bei abgeleiteten Wörtern die Präfixe und Suffixe abtrennen und somit das Kernmorphem liefern. Solche Regeln würden also z. B. bei der Wortform 'Unmenschlichkeit' das Präfix 'Un' und die Suffixkombination 'lichkeit' abtrennen und somit das Kernmorphem 'mensch' liefern. Der Lernkomponente des Algorithmus müßte in diesem Fall eine Menge von Wortpaaren vorgegeben werden, deren Erstkomponenten abgeleitete Wortformen sind und deren Zweitkomponenten die jeweiligen Kernmorpheme sind. Aus diesem Lernkorpus würde dann eine Menge von Anweisungen erlernt, die bei Ausführung durch die Anwendungskomponente des Algorithmus zu vorgegebenen abgeleiteten Wortformen die Kernmorpheme liefern.

   Eine derartige Anwendung des Algorithmus dürfte - wie bereits die Erlernung und Simulation morphologischer Ableitungsregeln - bei der Erhöhung des Recall in linguistisch orientierten Informationssystemen von Interesse sein.

6. Schließlich ließe sich der Algorithmus noch zur Erlernung von Flexionsregeln verwenden. Unklar ist hier allerdings, welchem übergeordneten Zweck eine solche Anwendung dienen könnte. Als Lernkorpus müßte dann eine Menge von Wortpaaren vorgegeben werden, deren Erstkomponenten Grundformen sind und deren Zweitkomponenten entsprechende Flexionsformen dieser Grundformen sind.

   Bei unserer experimentellen Arbeit mit PRISM haben wir versuchsweise einmal ein Lernkorpus erstellt, dessen Quellstrings Grundformen englischer Substantive waren und dessen Zielstrings die entsprechenden Pluralformen waren. Da dieses Lernkorpus für die regelhafteren Muster der Pluralbildung annähernd repräsentativ war, konnte erwartet werden, daß PRISM's Lernkomponente ausgehend von diesem Korpus die wichtigsten Regeln zur Pluralbildung englischer Substantive im wesentlichen erlernt. Für Ausnahmebildungen, wie z. B. 'oxen' zu 'ox' <Ochse> oder 'teeth' zu 'tooth' <Zahn> konnte die Erlernung von Derivationsanweisungen dagegen nicht erwartet werden, da das Lernkorpus keine entsprechenden Bei-

spiele enthielt. Das Lernkorpus bestand aus den Wortpaaren:

```
('assembly'   <Versammlung>, 'assemblies')
('bath'       <Bad>,         'baths')
('box'        <Kiste>,       'boxes')
('boy'        <Junge>,       'boys')
('bus'        <Bus>,         'buses')
('bush'       <Busch>,       'bushes')
('buzz'       <Gesumm>,      'buzzes')
('calf'       <Kalb>,        'calves')
('copy'       <Kopie>,       'copies')
('cry'        <Schrei>,      'cries')
('door'       <Tür>,         'doors')
('field'      <Feld>,        'fields')
('house'      <Haus>,        'houses')
('knife'      <Messer>,      'knives')
('lady'       <Dame>,        'ladies')
('mother'     <Mutter>,      'mothers')
('switch'     <Schalter>,    'switches')
('university' <Universität>, 'universities')
```

Ausgehend von diesem Lernkorpus ermittelte PRISM's Lernkomponente die präfixorientierte Substitutionsanweisung

'' → ''/# __

sowie die folgenden suffixorientierten Substitutionsanweisungen, die erwartungsgemäß im wesentlichen die wichtigsten Regeln zur Pluralbildung englischer Substantive wiedergeben:

'f' → 'ves'/__#
'fe' → 'ves'/__#
'y' → 'ies'/('d'|'l'|'p'|'r'|'t') __#
'' → 'es'/('ch'|'sh'|'s'|'x'|'z') __#
'' → 's'/__#

Lediglich die Kontextbedingung der Anweisung

'y' → 'ies'/('d'|'l'|'p'|'r'|'t') __#

ist noch nicht vollständig. Hier fehlen weitere 'konsonantische' Buchstaben, wie z. B. 'c' (Beispiel: 'agency' <Agentur> <==> 'agencies') und 'g' (Beispiel: 'elegy' <Elegie> <==> 'elegies'). Diese linguistische Inadäquatheit der Anweisung ist nicht dem Lernalgorithmus anzulasten. Vielmehr ist sie darauf zurückzuführen, daß das vorgegebene Lernkorpus für die erforderliche Kontextbedingung nicht repräsentativ ist.

Das Beispiel dürfte dennoch ausreichend verdeutlichen,

daß Flexionsregeln maschinell erlernt werden können.

Abschließend muß betont werden, daß für den Erfolg der Anwendungsmöglichkeiten 1, 4, 5 und 6 die Regelhaftigkeit der betrachteten morphologischen Beziehungen und die Repräsentativität der vorgegebenen Lernkorpora Voraussetzung ist. Genauer: Die morphologische Beziehung muß zunächst auf graphischer Ebene ausreichend regelhaft sein und, falls dies gegeben ist, muß auch ein repräsentatives Lernkorpus vorgegeben werden. Je geringer Regelhaftigkeit und Repräsentativität sind, desto größer dürfte i. allg. der erforderliche Aufwand an intellektueller Postedition der maschinell gewonnenen Ergebnisse sein. Weist eine morphologische Beziehung nur geringe Regelhaftigkeit auf, so sollte anstatt des Algorithmus zur Erlernung und Simulation morphologischer Ableitungsregeln besser ein maschinelles Lexikon verwendet werden.

9 UNZULÄNGLICHKEITEN UND WEITERENTWICKLUNGSMÖGLICHKEITEN VON PRISM

In diesem Abschlußkapitel möchten wir einige Unzulänglichkeiten des Systems PRISM aufzeigen, für die wir entweder keine Abhilfe finden konnten oder die wir aufgrund der zeitlichen Begrenzung unseres Vorhabens nicht beheben konnten. Soweit möglich sollen entsprechende Verbesserungsmöglichkeiten genannt werden, die u. U. zu einem effizienteren und anwendungsorientierten System führen können. Abschließend soll eine wesentliche Alternative zu dem PRISM zugrunde liegenden Basismodell (vgl. Abschnitt 1.1) diskutiert werden.

- Die Verwendung von Blockierungsanweisungen scheint linguistisch nicht zu rechtfertigen zu sein. Jedenfalls konnten wir kaum eine morphologische Ableitungsbeziehung finden, bei der sich *regelhaft* mit Impedienten angeben läßt, wann zu einem Quellstring kein Zielstring zu generieren ist. Es wäre wahrscheinlich sinnvoller, wenn PRISM's Lernkomponente diejenigen Quellstrings eines Lernkorpus, die einen leeren Zielstring haben, in eine Ausnahmenliste schreibt, anstatt zu ihnen Blockierungsanweisungen zu erlernen. Die Anwendungskomponente würde dann vor Anwendung der Substitutionsanweisungen auf einen Quellstring immer erst diese Ausnahmenliste durchsuchen. Falls der Quellstring in der Liste auftritt, würde kein Zielstring erzeugt, ansonsten würde versucht, die Substitutionsanweisungen anzuwenden.
- Die eingeführten Substitutionsanweisungen sind für die Beschreibung graphischer Veränderungen am Wort*anfang* und/oder am Wort*ende* konzipiert. Für die Beschreibung von Veränderungen im Wort*inneren*, die bei morphologischen Ableitungen allerdings auch seltener auftreten, sind sie nicht gut geeignet. Dies sei an einem Beispiel verdeutlicht. Nehmen wir an, in einem Lernkorpus zur Motion im Deutschen käme das Wortpaar

('Storch', 'Störchin')

vor. Dann würde PRISM's Lernkomponente zu diesem Wortpaar neben der durchaus zufriedenstellenden suffixorientierten Substitutionsanweisung

'' → 'in'/__#

die präfixorientierte Substitutionsanweisung

'Sto' → 'Stö'/#__

erlernen (Von Kontextbedingungen, die u. U. aufgrund der Opposition zu anderen an der Motion beteiligten Bildungsmustern eingeführt werden müßten, soll hier abgesehen werden). Die präfixorientierte Substitutionsanweisung beschreibt lediglich die Umlautung des Stammvokals. Zwar gibt sie die *in der vorgegebenen Beschreibungssprache kürzestmögliche* Darstellung dieser Veränderung (i. U. etwa zu 'Stor' → 'Stör'/#__), jedoch ist diese Darstellung aus linguistischer Sicht keineswegs zufriedenstellend: Strenggenommen muß nämlich nicht 'Sto' durch 'Stö' ersetzt werden, sondern lediglich 'o' durch 'ö'. Wie in diesem Beispiel, so geben die Substitutionsanweisungen in den meisten Fällen, in denen sich das Wortinnere bei einer Ableitung verändert und der Wortanfang und/oder das Wortende konstant bleibt, eine aus linguistischer Sicht redundante Beschreibung. Eine Lösung dieses Problems, etwa in Form einer neuen Beschreibungssprache, deren Ausdrücke auch maschinell erlernt werden können, haben wir bisher nicht gefunden.

- Bei der Weiterentwicklung von PRISM für Anwendungen im Rahmen linguistisch orientierter Informationssysteme ist es möglicherweise sinnvoll, außer mit Derivationsanweisungen auch mit einem Lexikon der Ausnahmen zu arbeiten. Wie wir bei der Überprüfung von Forderung 2 (vgl. Abschnitt 7) gesehen hatten, tritt bei Vorgabe nicht repräsentativer Lernkorpora häufig der Fehler auf, daß die Anwendungskomponente zu Quellstrings, die Ausnahmemustern angehören, mit den erlernten Derivationsanweisungen fälschlicherweise nach einem regelhafteren Bildungsmuster einen Zielstring erzeugt. Dieser Fehler läßt sich eher vermeiden, wenn das Lernkorpus nur aus Wortpaaren zusam-

mengestellt wird, die regelhafteren Bildungsmustern angehören, so daß die erlernten Derivationsanweisungen ausschließlich die regelhafteren Bildungsmuster beschreiben. Mit den Ausnahmebildungen sollte dagegen ein möglichst vollständiges Lexikon der Ausnahmen erstellt werden, das zu jedem, einem Ausnahmemuster angehörenden Quellstring den entsprechenden, unregelmäßig gebildeten Zielstring enthält. Die Anwendungskomponente von PRISM müßte dann vor Anwendung der Derivationsanweisungen auf einen vorgegebenen Quellstring stets zunächst das Lexikon auf den Quellstring durchsuchen. Ist der Quellstring im Lexikon enthalten, so wird der dort verzeichnete, unregelmäßig gebildete Zielstring als Resultat ausgegeben. Steht der Quellstring dagegen nicht im Lexikon, so erzeugt die Anwendungskomponente den Zielstring nach einem regelhaften Muster mit den erlernten Derivationsanweisungen.

- Die Alltagserfahrung und mehrere psycholinguistische Modelle des Spracherwerbs legen ein anderes Basismodell eines Algorithmus zur Erlernung und Simulation morphologischer Ableitungsregeln nahe, als wir es in dieser Arbeit zugrunde gelegt haben (vgl. Abb. 1.1). Beispielsweise entwickeln Bruce L. Derwing und William J. Baker in Orientierung an McCawley (1968) das folgende Modell des Erstspracherwerbs:

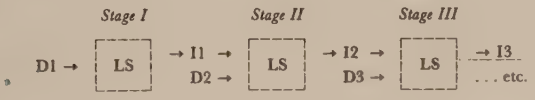


Abb. 9.1 (aus Derwing/Baker 1977: 93)

"In this still grossly simplified model, language acquisition is viewed in terms of a sequence of discrete developmental stages. At the initial stage it is assumed that a certain highly restricted body of linguistic (and other) data (D1) provide the input to the model, whose basic component consists of a set of particular native capacities or learning strategies (LS) which the child employs to infer, extract, and store some specific body of tentative information about the structure of his language (I1). At each subsequent stage it is this stored information, together with any new or repeated data (D2), which provides the potential input for the next stage, the original data no longer being available as such. It is also

possible, and even quite likely, that the child may periodically change his own basic strategies for coming to grips with his language, in which case it would be appropriate to distinguish the LSs which appear in the successive stages as well." (Derwing/Baker 1977: 93f.)

Ein ähnliches Modell legt auch Volker Beeh (1981: 98f.) vor.

Bei Orientierung am Modell von Derwing/Baker ergäbe sich etwa das in Abbildung 9.2 wiedergegebene Basismodell eines Algorithmus zur Erlernung und Simulation morphologischer Ableitungsregeln.

PRISM's Lernkomponente ist nicht in der Lage, eine früher erlernte, nicht adäquate Regelmenge aus der Wissensbasis mit einer aus weiteren Beispielwortpaaren erlernten Regelmenge zu einer neuen, adäquateren Menge zu synthetisieren. Vielmehr lassen sich mit PRISM adäquatere Regelmengen nur gewinnen, indem der Lernkomponente ein verbessertes Lernkorpus vorgegeben wird, aus dem sie dann eine völlig neue Regelmenge erlernt; zuvor erlernte Regeln werden bei diesem Lernprozess völlig vernachlässigt. Weiterhin ist PRISM's Lernstrategie starr vorgegeben. Ein System mit der in Abbildung 9.2 wiedergegebenen Grundstruktur wiese diese beiden Einschränkungen nicht auf:

- Die Lernkomponente wäre in der Lage, früher erlernte Regeln mit Regeln, die aus zusätzlichen Beispielwortpaaren erlernt wurden, zu einer neuen Regelmenge zu synthetisieren.
- Ein Fehleranalysator und Lernkomponentenmodifizierer würde vom Benutzer erkannte und gemeldete Fehler, die bei der Anwendung erlernter Regeln aufgetreten sind, evaluieren. Weiterhin würde er Inkonsistenzen zwischen einer bereits erlernten Regelmenge und zusätzlich vorgegebenen Lernbeispielen ermitteln.[1] In Abhängigkeit von seinen Evaluierungsergebnissen könnte er dann - soweit erforderlich - die Lernstrategien der Lernkomponente modifizieren.

1 Eine Inkonsistenz läge beispielsweise vor, wenn mit den bereits erlernten Regeln zu einem Quellstring der zusätzlich vorgegebenen Beispielwortpaare ein anderer Zielstring erzeugt würde, als er dem Quellstring im Beispielwortpaar beigegeben ist.

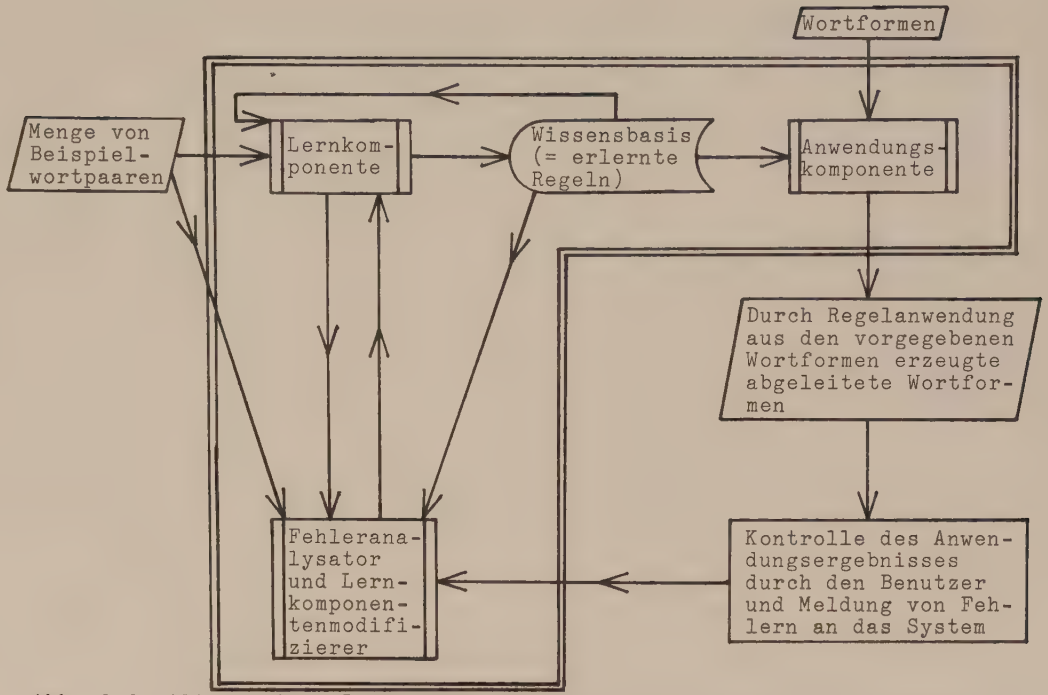


Abb. 9.2: Alternatives Basismodell für einen Algorithmus zur Erlernung und Simulation morphologischer Ableitungsregeln.

Trotz dieser - zunächst positiv erscheinenden - Eigenschaften haben wir PRISM nicht das in Abbildung 9.2 wiedergegebene Basismodell zugrunde gelegt, sondern das von Abbildung 1.1, und zwar aus folgenden Gründen:

- Mit PRISM sollte kein Modell für bestimmte menschliche Verhaltensweisen oder für bestimmte kognitive Prozesse beim Erstspracherwerb geliefert werden. Vielmehr sollte Grundlagenforschung für ein anwendungsorientiertes System geleistet werden. Es bestand daher kein Zwang, psycholinguistische Modelle zum Erstspracherwerb bei der Entwicklung von PRISM zu berücksichtigen und somit das Basismodell in Abbildung 9.2 zu wählen, das sich an einem psycholinguistischen Modell orientiert.
- Für ein anwendungsorientiertes System scheint das in Abbildung 9.2 angegebene Basismodell keine Vorteile gegenüber dem von uns gewählten Basismodell zu haben: Die Entwicklung und Programmierung einer Lernkomponente, die alte Regelmengen mit neuen Regelmengen synthetisieren kann, ist äußerst aufwendig, da es sich um ein logisch sehr komplexes Gebilde handelt. Das gleiche gilt für den Fehleranalysator und Lernkomponentenmodifizierer, bei dem noch nicht einmal klar ist, in welcher Hinsicht er in Abhängigkeit von aufgetretenen Fehlern oder Inkonsistenzen die Strategien der Lernkomponente modifizieren soll. Es ist nicht sicher, ob ein solches System in konkreten Anwendungssituationen irgendwelche Vorteile gegenüber PRISM hätte, die den entschieden höheren Entwicklungsaufwand rechtfertigen könnten. In seiner Benutzerfreundlichkeit dürfte sich das System kaum von PRISM unterscheiden. Wahrscheinlich würde die Lernphase rechenzeitintensiver sein als bei PRISM: Denn zum einen müßte grundsätzlich der Fehleranalysator und Lernkomponentenmodifizierer zu Beginn jedes Lernlaufs die Beispielwortpaare mit bereits existierenden Regeln auf Inkonsistenzen vergleichen und ggf. die Lernkomponente modifizieren. Zum anderen wäre die Lernkomponente um einen voraussichtlich sehr rechenzeitintensiven Synthesemechanismus erweitert. Schließlich weist nichts darauf hin, daß - bei Verwendung

der von PRISM benutzten Beschreibungssprache - die erlernten Regelmengen irgendwelche Vorteile gegenüber den von PRISM erlernten Regelmengen haben könnten.

10 ANHANG

10.1 PRISM-Quellprogramm

```
/*-------------------------------------------------------------*/
/* NAME:  P R I S M                                            */
/*                                                             */
/* VERSION: 1                                                  */
/*                                                             */
/* DATE OF LAST RELEASE: 15 AUG. 1983                          */
/*                                                             */
/* AUTHOR: K. WOTHKE                                           */
/*         INSTITUT FUER KOMMUNIKATIONS-                       */
/*         FORSCHUNG UND PHONETIK                              */
/*         UNIVERSITY OF BONN                                  */
/*         POPPELSDORFER ALLEE 47                              */
/*         53 BONN 1                                           */
/*         GERMANY (WEST)                                      */
/*                                                             */
/* FUNCTION:                                                   */
/* =========                                                   */
/*                                                             */
/* THIS IS ONLY A BRIEF DESCRIPTION OF THE FUNCTION OF         */
/* PRISM. IT IS NOT INTENDED AS A USER'S GUIDE TO PRISM.       */
/* FOR A DETAILED DESCRIPTION OF PRISM SEE THE PREREQUI-       */
/* SITE DESCRIPTIONS SPECIFIED BELOW.                          */
/*    PRISM (= PROGRAM FOR THE INFERENCE AND SIMULATION        */
/* OF MORPHOLOGICAL RULES) IS A PROGRAM FOR THE TREATMENT      */
/* OF CERTAIN LINGUISTIC PROBLEMS. IT CONSISTS OF 2 MAIN       */
/* COMPONENTS:                                                 */
/*  - ONE COMPONENT, THE SO-CALLED LEARNING-ALGORITHM,         */
/*    INFERS FROM LINGUISTIC EXAMPLES, WHICH ARE GIVEN BY      */
/*    THE USER OF PRISM, SO-CALLED SETS OF DERIVATION-         */
/*    STATEMENTS. THESE SETS ARE ALGORITHMIC DESCRIPTIONS      */
/*    OF CERTAIN LINGUISTIC PHENOMENA IN THE GIVEN EXAM-       */
/*    PLES.                                                    */
/*  - THE OTHER MAIN COMPONENT IS AN ALGORITHM FOR THE         */
/*    APPLICATION OF DERIVATION-STATEMENTS WHICH WERE IN-      */
/*    FERED BY THE LEARNING-ALGORITHM.                         */
/* A THIRD, MARGINAL COMPONENT OF PRISM WRITES DERIVA-         */
/* TION-STATEMENTS WHICH WERE INFERED BY THE LEARNING-AL-      */
/* GORITHM ONTO THE SYSPRINT-FILE.                             */
/*    BY SPECIFYING APPROPRIATE OPTIONS IN THE DATA SET        */
/* ALLOCATED TO SYSIN (SEE BELOW) THE USER DETERMINES          */
/* WHICH COMPONENT OF PRISM IS TO BE ACTIVATED. THE            */
/* OPTIONS, WHICH CAN BE USED TO CONTROL PRISM, ARE:           */
/*  LEARN         - THE LEARNING-ALGORITHM IS TO BE ACTI-      */
/*                  VATED.                                     */
/*  APPLY         - THE ALGORITHM FOR THE APPLICATION OF       */
/*                  DERIVATION-STATEMENTS IS TO BE ACTI-       */
/*                  VATED.                                     */
/*  PRINT_STMTS   - THE ALGORITHM FOR THE OUTPUT OF DERI-      */
/*                  VATION-STATEMENTS ONTO THE SYSPRINT-       */
/*                  FILE IS TO BE ACTIVATED.                   */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/*  PRINT        - THE RESULT OF THE ACTIVATED ALGORITHM       */
/*                 IS TO BE WRITTEN ALSO ONTO THE              */
/*                 SYSPRINT-FILE.                              */
/*  NOPRINT      - THE RESULT OF THE ACTIVATED ALGORITHM       */
/*                 SHALL NOT BE WRITTEN ONTO THE SYSPRINT-     */
/*                 FILE.                                       */
/*  STATISTICS   - DERIVATION-STATEMENTS ARE TO BE WRIT-       */
/*                 TEN  W I T H  STATISTICAL MESSAGES ON-      */
/*                 TO THE SYSPRINT-FILE.                       */
/*  NOSTATISTICS - DERIVATION-STATEMENTS ARE TO BE WRIT-       */
/*                 TEN  W I T H O U T  STATISTICAL MES-        */
/*                 SAGES ONTO THE SYSPRINT-FILE.               */
/* THE VALID COMBINATIONS OF THESE OPTIONS ARE DEFINED BY      */
/* THE FOLLOWING SYNTAX:                                       */
/*                                                             */
/*     +-              +-          +-                -+ -+ -+  */
/*     |               |           | STATISTICS      |  |  |  */
/*     |               |           | ----------      |  |  |  */
/*     |               | PRINT     |                 |  |  |  */
/*     |               | -----     |                 |  |  |  */
/*     | LEARN         |           | NOSTATISTICS    |  |  |  */
/*     |               |           +-                -+  |  |  */
/*     |               | NOPRINT                         |  |  */
/*     |               +-                               -+  |  */
/*     |                                                     |  */
/*     |               +-        -+                         |  */
/*     |               | PRINT    |                         |  */
/*     | APPLY         |          |                         |  */
/*     | -----         |          |                         |  */
/*     |               | NOPRINT  |                         |  */
/*     |               | -------  |                         |  */
/*     |               +-        -+                         |  */
/*     |                                                     |  */
/*     |               +-              -+                    |  */
/*     |               | STATISTICS     |                    |  */
/*     |               | ----------     |                    |  */
/*     | PRINT_STMTS   |                |                    |  */
/*     |               | NOSTATISTICS   |                    |  */
/*     +-              +-              -+                   -+  */
/*                                                             */
/* (OPTIONS IN BRACKETS ARE OPTIONAL. OPTIONS WHICH ARE        */
/* IN THE SAME COLUMN ARE ALTERNATIVE. IF OPTIONAL OP-         */
/* TIONS ARE NOT SPECIFIED, THE UNDERLINED OPTIONS ARE         */
/* USED BY PRISM AS DEFAULT OPTIONS. IF SUCH A DEFAULT         */
/* OPTION HAS DEPENDENT OPTIONS YOU CAN NEVERTHELESS           */
/* SPECIFY THE DEPENDENT OPTIONS.)                             */
/*    IN DETAIL, THE COMPONENTS OF PRISM HAVE THE FOLLOW-      */
/* ING FUNCTIONS:                                              */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/*    INPUT TO THE  L E A R N I N G - A L G O R I T H M        */
/* IS A SET OF LINGUISTIC EXAMPLES, CALLED LEARNING-COR-       */
/* PUS. THIS LEARNING-CORPUS MUST BE GIVEN IN THE DATA         */
/* SET ALLOCATED TO CORPUS (SEE BELOW). THE EXAMPLES MUST      */
/* BE ORDERED PAIRS OF STRINGS. THE FIRST STRING OF EACH       */
/* PAIR IS CALLED  S O U R C E, THE SECOND STRING IS            */
/* CALLED  T A R G E T. SOURCE AND TARGET MUST AGREE WITH      */
/* THE FOLLOWING CONVENTIONS:                                  */
/*  - SOURCE AND TARGET  S H O U L D  EACH REPRESENT A         */
/*    WORD OF A NATURAL LANGUAGE.                              */
/*  - THE GRAPHIC DIFFERENCES BETWEEN SOURCE AND TARGET        */
/*    S H O U L D  APPEAR AT THE BEGINNING AND/OR THE END      */
/*    OF SOURCE AND TARGET. THEY SHOULD NOT APPEAR IN          */
/*    THEIR INTERIOR.                                          */
/*  - THE SOURCE OF EACH GIVEN PAIR MAY APPEAR ONLY ONCE       */
/*    AS A SOURCE IN A GIVEN LEARNING-CORPUS.                  */
/*  - THE TARGET MAY ALSO BE THE NULL STRING, I. E. THE        */
/*    STRING WITH THE LENGTH 0.                                */
/*  - SOURCE AND TARGET EACH MUST NOT BE LONGER THAN 30        */
/*    CHARACTERS.                                              */
/*  - SOURCE AND TARGET EACH MUST NOT CONTAIN THE EBCDIC-      */
/*    CHARACTERS WITH THE HEXADECIMAL VALUES 00 AND FF.        */
/*    FOR EACH LEARNING-CORPUS WHICH AGREES WITH THESE         */
/* FORMAL CONVENTIONS THE LEARNING-ALGORITHM INFERS A SET      */
/* OF DERIVATION-STATEMENTS, WHICH DESCRIBES ON A PURELY       */
/* GRAPHIC BASIS HOW THE TARGET OF EACH PAIR MAY BE GEN-       */
/* ERATED FROM THE SOURCE. IF THESE DERIVATION-STATEMENTS      */
/* ARE LATER APPLIED TO THE SOURCE OF AN ARBITRARY PAIR        */
/* FROM THE EXAMPLES, THEY GENERATE THE TARGET BELONGING       */
/* TO IT. IF FOR A GIVEN SOURCE A TARGET DOES NOT EXIST,       */
/* I. E., IF THE TARGET IS THE NULL STRING, THE DERIVA-        */
/* TION-STATEMENTS DO NOT GENERATE A TARGET FOR THIS           */
/* SOURCE.                                                     */
/*    THE LEARNING-ALGORITHM TRIES TO DETECT ANALOGIES IN      */
/* THE GENERATION OF THE TARGETS FROM THE SOURCES, AND IT      */
/* INCORPORATES THESE ANALOGIES INTO THE DERIVATION-           */
/* STATEMENTS. IF REALLY ANALOGIES EXIST, THE DERIVATION-      */
/* STATEMENTS THEREFORE CAN ALSO GENERATE CORRECT TARGETS      */
/* FOR SUCH SOURCES WHICH WERE NOT SPECIFIED IN THE            */
/* LEARNING-CORPUS. THIS SHALL BE ILLUSTRATED BY AN EXAM-      */
/* PLE FROM GERMAN: SUPPOSE, THE INPUT TO THE LEARNING-        */
/* ALGORITHM IS THE FOLLOWING LEARNING-CORPUS:                 */
/*                                                             */
/*   BEARBEITEN                    BEARBEITUNG                 */
/*   PRODUZIEREN                   PRODUKTION                  */
/*                                                             */
/* THEN THE INFERED DERIVATION-STATEMENTS DO NOT ONLY          */
/* GENERATE 'BEARBEITUNG' FROM 'BEARBEITEN' AND                */
/* 'PRODUKTION' FROM 'PRODUZIEREN', BUT E. G. ALSO             */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/* 'BENENNUNG' FROM 'BENENNEN', 'BETREUUNG' FROM               */
/* 'BETREUEN' AND 'DEDUKTION' FROM 'DEDUZIEREN'. THE           */
/* PROBABILITY WITH WHICH THE DERIVATION-STATEMENTS GEN-       */
/* ERATE CORRECT TARGETS FOR SUCH SOURCES WHICH DO NOT         */
/* BELONG TO THE LEARNING-CORPUS MAINLY DEPENDS ON 2 FAC-      */
/* TORS:                                                       */
/*  - THE REPRESENTATIVITY OF THE LEARNING-CORPUS FOR THE      */
/*    TYPE OF TARGET-GENERATION THE DERIVATION-STATEMENTS      */
/*    SHALL DESCRIBE.                                          */
/*  - THE REGULARITY OF THE TYPE OF TARGET-GENERATION ON       */
/*    THE GRAPHIC LEVEL.                                       */
/* IF A TYPE OF TARGET-GENERATION IS SUFFICIENTLY REGULAR      */
/* AND IF THE LEARNING-CORPUS IS SUFFICIENTLY REPRESENTA-      */
/* TIVE FOR THIS TYPE OF TARGET-GENERATION, THE LEARNING-      */
/* ALGORITHM GENERATES - IN THE FORM OF THE DERIVATION-        */
/* STATEMENTS - AN EFFICIENT ALGORITHM, WHICH CAN BE USED      */
/* TO GENERATE TARGETS FROM GIVEN SOURCES. THEREFORE THE       */
/* LEARNING-ALGORITHM OF PRISM MAY BE ESPECIALLY USEFUL        */
/* FOR THE AUTOMATIC GENERATION OF ALGORITHMS FOR              */
/*  - THE AUTOMATIC GENERATION OF DERIVATIVES (E. G. THE       */
/*    GENERATION OF NOUNS FROM VERBS, SUCH AS                  */
/*    'DISTRIBUTION' FROM 'DISTRIBUTE'),                       */
/*  - THE AUTOMATIC LEMMATIZATION                              */
/*  - AND THE AUTOMATIC GENERATION OF STEMS (SUCH AS THE       */
/*    GENERATION OF 'LES' FROM 'LESEN' IN GERMAN).             */
/*    THE DERIVATION-STATEMENTS INFERED BY THE LEARNING-       */
/* ALGORITHM ARE DIVIDED INTO 4 SUBCLASSES:                    */
/*  - P R E F I X A L - B L O C K I N G - S T A T E -          */
/*    M E N T S  P R E V E N T  THE GENERATION OF A TAR-       */
/*    GET IF THE SOURCE, TO WHICH THE STATEMENTS ARE AP-       */
/*    PLIED, B E G I N S  WITH A STRING X SPECIFIED IN         */
/*    ONE OF THE STATEMENTS.                                   */
/*    DEFINITION: PREFIXAL-BLOCKING-STATEMENTS ARE ALL         */
/*    AND ONLY THOSE STATEMENTS OF THE FORM                    */
/*                                                             */
/*       #X¬,                                                  */
/*                                                             */
/*    WHERE X IS A - POSSIBLY EMPTY - STRING.                  */
/*    SUCH A STATEMENT MEANS: IF A SOURCE BEGINS WITH THE      */
/*    STRING X, DO NOT GENERATE A TARGET.                      */
/*  - S U F F I X A L - B L O C K I N G - S T A T E -          */
/*    M E N T S  P R E V E N T  THE GENERATION OF A TAR-       */
/*    GET IF THE SOURCE, TO WHICH THE STATEMENTS ARE AP-       */
/*    PLIED,  E N D S  WITH A STRING X SPECIFIED IN ONE        */
/*    OF THE STATEMENTS.                                       */
/*    DEFINITION: SUFFIXAL-BLOCKING-STATEMENTS ARE ALL         */
/*    AND ONLY THOSE STATEMENTS OF THE FORM                    */
/*                                                             */
/*       X#¬,                                                  */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------*/
/*    WHERE X IS A - POSSIBLY EMPTY - STRING.            */
/*    SUCH A STATEMENT MEANS: IF A SOURCE ENDS WITH THE  */
/*    STRING X, DO NOT GENERATE A TARGET.                */
/*  - P R E F I X A L - S U B S T I T U T I O N -        */
/*    S T A T E M E N T S  C H A N G E  T H E  B E G I N -*/
/*    N I N G  OF A SOURCE IN ORDER TO GENERATE THE COR- */
/*    RESPONDING TARGET.                                 */
/*    DEFINITION: PREFIXAL-SUBSTITUTION-STATEMENTS ARE   */
/*    ALL AND ONLY THOSE STATEMENTS OF THE FORM          */
/*                                                       */
/*       X -> Y/#__(Z(1)| ... |Z(I)| ... |Z(N)),         */
/*                                                       */
/*    WHERE                                              */
/*     - X AND Y ARE - POSSIBLY EMPTY - STRINGS,         */
/*     - Z(1), ..., Z(N) ARE (NOT EMPTY) STRINGS,        */
/*     - N >= 0.                                         */
/*    SUCH A STATEMENT MEANS: IF A SOURCE BEGINS WITH THE*/
/*    STRING X AND IF IMMEDIATELY ON THE RIGHT OF X FOL- */
/*    LOWS THE STRING Z(1) OR ... OR Z(I) OR ... OR Z(N),*/
/*    THEN SUBSTITUTE X BY Y.                            */
/*  - S U F F I X A L - S U B S T I T U T I O N -        */
/*    S T A T E M E N T S  C H A N G E  THE  E N D  OF A */
/*    SOURCE IN ORDER TO GENERATE THE CORRESPONDING TAR- */
/*    GET.                                               */
/*    DEFINITION: SUFFIXAL-SUBSTITUTION-STATEMENTS ARE   */
/*    ALL AND ONLY THOSE STATEMENTS OF THE FORM          */
/*                                                       */
/*       X -> Y/(Z(1)| ... |Z(I)| ... |Z(N))__#,        */
/*                                                       */
/*    WHERE                                              */
/*     - X AND Y ARE - POSSIBLY EMPTY - STRINGS,         */
/*     - Z(1), ..., Z(N) ARE (NOT EMPTY) STRINGS,        */
/*     - N >= 0.                                         */
/*    SUCH A STATEMENT MEANS: IF A SOURCE ENDS WITH THE  */
/*    STRING X AND IF IMMEDIATELY ON THE LEFT OF X IS THE*/
/*    THE STRING Z(1) OR ... OR Z(I) OR ... OR Z(N), THEN*/
/*    SUBSTITUTE X BY Y.                                 */
/*    A SET OF DERIVATION-STATEMENTS INFERED BY THE      */
/* LEARNING-ALGORITHM IS ORDERED, I. E. THE APPLICATION  */
/* OF THE STATEMENTS TO A GIVEN SOURCE MUST BE TRIED IN A*/
/* F I X E D  SEQUENCE IN ORDER TO GENERATE A TARGET. AT */
/* FIRST THE APPLICATION OF THE BLOCKING-STATEMENTS MUST  */
/* BE TRIED. IF ONE OF THESE STATEMENTS IS APPLICABLE, NO */
/* TARGET IS GENERATED FOR THE SOURCE AND IT IS NOT TRIED */
/* TO APPLY ANY OTHER STATEMENT. OTHERWISE THE  F I R S T */
/* APPLICABLE PREFIXAL- AND THE  F I R S T  APPLICABLE    */
/* SUFFIXAL-SUBSTITUTION-STATEMENT MUST BE DETERMINED AND */
/* BOTH MUST CONCURRENTLY BE APPLIED TO THE SOURCE,       */
/* WHEREBY THE TARGET IS GENERATED.                      */
/*    THE LEARNING-ALGORITHM STORES THE GENERATED SET OF */
/*-------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/* DERIVATION-STATEMENTS IN THE DATA SET ALLOCATED TO          */
/* STMTFLE (SEE BELOW), SO THAT IT MAY BE PROCESSED DUR-       */
/* ING A LATER ACTIVATION OF THE ALGORITHM FOR THE APPLI-      */
/* CATION OR THE ALGORITHM FOR THE OUTPUT OF DERIVATION-       */
/* STATEMENTS. THE STATEMENTS ARE STORED IN THE DATA SET       */
/* IN A FORM WHICH IS NOT IDENTICAL WITH THAT GIVEN IN         */
/* THE ABOVE DEFINITIONS OF THE STATEMENTS. THE FORM           */
/* GIVEN IN THE DEFINITIONS IS USED BY PRISM ONLY, WHEN        */
/* PRISM WRITES DERIVATION-STATEMENTS ONTO THE SYSPRINT-       */
/* FILE.  I N S I D E  OF PRISM AND IN THE DATA SET ALLO-      */
/* CATED TO STMTFLE, THE DERIVATION-STATEMENTS ARE ORGAN-      */
/* IZED IN A WAY WHICH ALLOWS THEIR EASY AND FAST AUTO-        */
/* MATIC TREATMENT.                                            */
/*    INPUT TO THE  A L G O R I T H M  F O R  T H E            */
/* A P P L I C A T I O N  O F  D E R I V A T I O N -           */
/* S T A T E M E N T S  IS                                     */
/*  1. A SET OF DERIVATION-STATEMENTS INFERED BY THE           */
/*     LEARNING-ALGORITHM. THIS SET OF DERIVATION-STATE-       */
/*     MENTS MUST BE GIVEN IN THE DATA SET ALLOCATED TO        */
/*     STMTFLE (SEE BELOW).                                    */
/*  2. ONE OR MORE SOURCES, FOR WHICH THE ALGORITHM SHALL      */
/*     GENERATE TARGETS. THE SOURCES MUST BE GIVEN IN THE      */
/*     DATA SET ALLOCATED TO CORPUS (SEE BELOW).               */
/* THE ALGORITHM APPLIES THE DERIVATION-STATEMENTS TO THE      */
/* SOURCES, THEREBY GENERATES THE TARGETS AND STORES THE       */
/* GENERATED TARGETS WITH THE CORRESPONDING SOURCES IN         */
/* THE DATA SET ALLOCATED TO CORPUS. THE ALGORITHM TRIES       */
/* TO APPLY THE STATEMENTS BY A METHOD WHICH IS NOT IDEN-      */
/* TICAL, BUT (IN ITS RESULT) EQUIVALENT TO THE STRATEGY       */
/* OUTLINED ABOVE IN THE DESCRIPTION OF THE LEARNING-AL-       */
/* GORITHM.                                                    */
/*    THE  A L G O R I T H M  F O R  T H E  O U T P U T        */
/* O F  D E R I V A T I O N - S T A T E M E N T S  READS       */
/* FROM THE DATA SET ALLOCATED TO STMTFLE (SEE BELOW)          */
/* DERIVATION-STATEMENTS WHICH WERE INFERED DURING A FOR-      */
/* MER ACTIVATION OF THE LEARNING-ALGORITHM AND WRITES         */
/* THEM ONTO THE SYSPRINT-FILE. FOR THE OUTPUT THE DERI-       */
/* VATION-STATEMENTS ARE TRANSFORMED FROM THEIR INTERNAL       */
/* REPRESENTATION INTO THE FORM USED IN THE ABOVE DEFINI-      */
/* TIONS.                                                      */
/*                                                             */
/* PREREQUISITE DESCRIPTIONS:                                  */
/* ==========================                                  */
/*                                                             */
/*  WOTHKE, K. (1984): PRISM USER'S GUIDE. (UNPUBLISHED        */
/*     PAPER.)                                                 */
/*  WOTHKE, K. (IN PREP.): MASCHINELLE ERLERNUNG UND SI-       */
/*     MULATION MORPHOLOGISCHER ABLEITUNGSREGELN. (DOC-        */
/*     TORAL DISSERTATION, UNIVERSITY OF BONN.)                */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/* USED FILES:                                                 */
/* ===========                                                 */
/*                                                             */
/*  FILENAME  FUNCTION                        ATTRIBUTES       */
/*  --------  --------                        ----------       */
/*                                                             */
/*  CORPUS    THE DATA SET ALLOCATED TO       DATA SET ORG-    */
/*            CORPUS MUST CONTAIN             ANIZATION:       */
/*             - THE LEARNING-CORPUS IF       PHYSICAL SE-     */
/*               THE LEARNING-ALGORITHM       QUENTIAL;        */
/*               OF PRISM IS ACTIVATED.       RECORD FORMAT:   */
/*               BYTE 11-40 OF EACH REC-      FIXED BLOCKED;   */
/*               ORD MUST CONTAIN (LEFT       LOGICAL RECORD   */
/*               ADJUSTED) A SOURCE, BYTE     LENGTH: 80       */
/*               41-70 MUST CONTAIN (LEFT     BYTE.            */
/*               ADJUSTED) THE CORRE-                          */
/*               SPONDING TARGET.                              */
/*             - SOURCES FOR WHICH TAR-                        */
/*               GETS ARE TO BE GENERATED                      */
/*               IF THE ALGORITHM FOR THE                      */
/*               APPLICATION OF DERIVA-                        */
/*               TION-STATEMENTS IS ACTI-                      */
/*               VATED. THE SOURCES MUST                       */
/*               BE ENTERED (LEFT ADJUST-                      */
/*               ED) IN BYTE 11-40 OF                          */
/*               EACH RECORD. FOR EACH                         */
/*               SOURCE PRISM STORES THE                       */
/*               GENERATED TARGET LEFT                         */
/*               ADJUSTED IN BYTE 41-70                        */
/*               OF THE RECORD WHICH CON-                      */
/*               TAINS THE SOURCE.                             */
/*            THIS FILE IS NOT USED BY                         */
/*            THE ALGORITHM FOR THE OUT-                       */
/*            PUT OF DERIVATION-STATE-                         */
/*            MENTS.                                           */
/*                                                             */
/*  STMTFLE   THE DATA SET ALLOCATED TO       DATA SET ORG-    */
/*            STMTFLE                         ANIZATION:       */
/*             - IS FILLED WITH DERIVA-       PHYSICAL SE-     */
/*               TION STATEMENTS WHEN THE     QUENTIAL;        */
/*               LEARNING-ALGORITHM IS        RECORD FORMAT:   */
/*               ACTIVATED.                   FIXED BLOCKED;   */
/*             - MUST CONTAIN DERIVATION-     LOGICAL RECORD   */
/*               STATEMENTS, WHICH WERE       LENGTH: 70       */
/*               WRITTEN INTO IT DURING A     BYTE.            */
/*               FORMER ACTIVATION OF THE                      */
/*               LEARNING-ALGORITHM, IF                        */
/*               THE ALGORITHM FOR THE                         */
/*               APPLICATION OR THE ALGO-                      */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/*  FILENAME   FUNCTION                        ATTRIBUTES      */
/*  --------   --------                        ----------      */
/*                                                             */
/*             RITHM FOR THE OUTPUT OF                         */
/*             DERIVATION-STATEMENTS IS                        */
/*             ACTIVATED.                                      */
/*                                                             */
/*  SYSIN      BYTE 1-50 OF THE FIRST REC-     STANDARD AT-    */
/*             CRD IN THE DATA SET ALLO-       TRIBUTES FOR    */
/*             CATED TO SYSIN MUST CONTAIN     THE SYSIN-FILE  */
/*             THE OPTIONS FOR THE CONTROL     OR:             */
/*             OF PRISM.                       DATA SET OR-    */
/*                                             GANIZATION:     */
/*                                             PHYSICAL SE-    */
/*                                             QUENTIAL;       */
/*                                             RECORD FORMAT:  */
/*                                             FIXED BLOCKED;  */
/*                                             LOGICAL RECORD  */
/*                                             LENGTH: 50      */
/*                                             BYTE.           */
/*                                                             */
/*  SYSPRINT   INTO THE DATA SET ALLOCATED     STANDARD AT-    */
/*             TO SYSPRINT PRISM WRITES        TRIBUTES FOR    */
/*             MESSAGES AND - EVENTUALLY       THE SYSPRINT-   */
/*             - RESULTS.                      FILE OR:        */
/*                                             DATA SET OR-    */
/*                                             GANIZATION:     */
/*                                             PHYSICAL SE-    */
/*                                             QUENTIAL;       */
/*                                             RECORD FORMAT:  */
/*                                             FIXED BLOCKED;  */
/*                                             LOGICAL RECORD  */
/*                                             LENGTH: 71      */
/*                                             BYTE.           */
/*                                                             */
/*  WRK1FLE    THE DATA SET ALLOCATED TO       DATA SET OR-    */
/*             WRK1FLE IS USED FOR INTER-      GANIZATION:     */
/*             MEDIATE STORAGE BY THE          PHYSICAL SE-    */
/*             LEARNING-ALGORITHM. THE AL-     QUENTIAL;       */
/*             GORITHM FOR THE APPLICATION     RECORD FORMAT:  */
/*             AND THE ALGORITHM FOR THE       FIXED BLOCKED;  */
/*             OUTPUT OF DERIVATION-STATE-     LOGICAL RECORD  */
/*             MENTS DO NOT USE THIS FILE.     LENGTH: 73      */
/*                                             BYTE.           */
/*                                                             */
/*  WRK2FLE    THE DATA SET ALLOCATED TO       DATA SET OR-    */
/*             WRK2FLE IS USED FOR INTER-      GANIZATION:     */
/*             MEDIATE STORAGE BY THE          PHYSICAL SE-    */
/*             LEARNING-ALGORITHM. THE AL-     QUENTIAL;       */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/*  FILENAME  FUNCTION                        ATTRIBUTES       */
/*  --------  --------                        ----------       */
/*                                                             */
/*            GORITHM FOR THE APPLICATION     RECORD FORMAT:   */
/*            AND THE ALGORITHM FOR THE       FIXED BLOCKED;   */
/*            OUTPUT OF DERIVATION-STATE-     LOGICAL RECORD   */
/*            MENTS DO NOT USE THIS FILE.     LENGTH: 190      */
/*                                            BYTE.            */
/*                                                             */
/*  WRK3FLE   THE DATA SET ALLOCATED TO       DATA SET OR-     */
/*            WRK3FLE IS USED FOR INTER-      GANIZATION:      */
/*            MEDIATE STORAGE BY THE          DIRECT ACCESS;   */
/*            LEARNING-ALGORITHM. THE AL-     RECORD FORMAT:   */
/*            GORITHM FOR THE APPLICATION     FIXED;           */
/*            AND THE ALGORITHM FOR THE       BLOCKSIZE: 190   */
/*            OUTPUT OF DERIVATION-STATE-     BYTE.            */
/*            MENTS DO NOT USE THIS FILE.                      */
/*                                                             */
/*    FURTHERMORE THE SORT-UTILITY SYNCSORT, WHICH IS          */
/* CALLED BY THE LEARNING-ALGORITHM OF PRISM VIA THE IN-       */
/* TERFACE PROGRAM PLISRTA, NEEDS THE FOLLOWING FILES:         */
/*            WRK1IN,   WRK2IN,                                */
/*            WRK1OUT,  WRK2OUT,                               */
/*            WRK1WK01, WRK2WK01,                              */
/*            WRK1WK02, WRK2WK02,                              */
/*            WRK1WK03, WRK2WK03.                              */
/*  - WRK1IN AND WRK1OUT MUST BE ALLOCATED TO THE SAME         */
/*    DATA SET AS WRK1FLE (S. A.).                             */
/*  - WRK2IN AND WRK2OUT MUST BE ALLOCATED TO THE SAME         */
/*    DATA SET AS WRK2FLE (S. A.).                             */
/*  - WRK1WK01, WRK1WK02, WRK1WK03, WRK2WK01, WRK2WK02         */
/*    AND WRK2WK03 MUST BE ALLOCATED TO TEMPORARY DATA         */
/*    SETS, WHICH ARE USED FOR INTERMEDIATE STORAGE BY         */
/*    SYNCSORT. ONLY THE SPACE REQUIREMENTS OF THESE DATA      */
/*    SETS AND THE UNIT, ON WHICH THEY ARE TO RESIDE,          */
/*    MUST BE SPECIFIED.                                       */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/* NAME: MONITOR                                               */
/*                                                             */
/* FUNCTION:                                                   */
/* MONITOR IS THE MAIN-PROCEDURE OF PRISM. IN DEPENDENCE       */
/* OF THE OPTIONS SPECIFIED FOR THE EXECUTION OF PRISM IT      */
/* ACTIVATES                                                   */
/*  - THE LEARNING-ALGORITHM,                                  */
/*  - THE ALGORITHM FOR THE APPLICATION OF DERIVATION-         */
/*    STATEMENTS                                               */
/*  - OR THE ALGORITHM FOR THE OUTPUT OF DERIVATION-           */
/*    STATEMENTS.                                              */
/* FURTHERMORE MONITOR COORDINATES THE ACTIVATION OF THE       */
/* SUB-ALGORITHMS WHICH CONSTITUTE THESE 3 ALGORITHMS.         */
/*    MONITOR ALSO WRITES THE FRONT PAGE AND THE LAST          */
/* PAGE OF THE PRISM-OUTPUT ONTO THE SYSPRINT-FILE. IF         */
/* NECESSARY, MONITOR WRITES ONTO THE SYSPRINT-FILE ER-        */
/* ROR-MESSAGES CONCERNING ONE OF THE FILES USED IN            */
/* PRISM.                                                      */
/*                                                             */
/* FILE:                                                       */
/*  OUTPUT: SYSPRINT.                                          */
/*-------------------------------------------------------------*/

MONITOR: PROC OPTIONS(MAIN) REORDER;
DCL      1 STMT(*)              CTL EXT,
          2 X_Z                 CHAR(30) VAR,
          2 LGT_X               BIN FIXED(15),
          2 Y                   CHAR(30) VAR,
          2 FREQ_SRC_OR_LLCXZ   BIN FIXED(31),

         MODE                   CHAR(11) VAR,
         DAT(3)                 CHAR(2),
         TIM(3)                 CHAR(2),
         PRINT                  BIT(1),
         STATISTICS             BIT(1),
         MAXSTNO                BIN FIXED(15) EXT,
         PAGENO                 BIN FIXED(15) EXT,
         RETURN_CODE            BIN FIXED(31) INIT(0),
         (DATE,STRING,TIME)     BUILTIN,
         CORPUS                 FILE RECORD SEQI,
         STMTFLE                FILE RECORD SEQI,
         SYSIN                  FILE STREAM INPUT,
         SYSPRINT               FILE STREAM PRINT,
         WRK1FLE                FILE RECORD SEQI,
         WRK2FLE                FILE RECORD SEQI,
         WRK3FLE                FILE RECORD KEYED
                                   ENV(REGIONAL(1)),
         GETOPTN                ENTRY(CHAR(11) VAR,BIT(1),
                                   BIT(1),BIN FIXED(31)),
         CHKCRPS                ENTRY(BIN FIXED(31)),
         DISCOV                 ENTRY(BIN FIXED(31)),
         STMTOUT                ENTRY(BIT(1)),
         TCDSET                 ENTRY,
         PRODSET                ENTRY(CHAR(11) VAR,
                                   BIN FIXED(31)),
```

```
        APPLY                   ENTRY,
        LISTOUT                 ENTRY;

/*----------------------------------------------------------*/
/* DEFINITION OF THE MAXIMUM NUMBER OF DERIVATION-STATE-    */
/* MENTS (= MAXSTNO) PRISM CAN PROCESS AND ALLOCATION OF    */
/* THE STRUCTURE-ARRAY STMT, WHICH IS TO CONTAIN THE DER-   */
/* IVATION-STATEMENTS, WITH THIS NUMBER AS UPPER BOUND.     */
/* (THE VALUE ASSIGNED TO MAXSTNO MUST BE GREATER THAN OR   */
/* EQUAL TO 1 AND LESS THAN OR EQUAL TO 32767.)             */
/*----------------------------------------------------------*/

        MAXSTNO = 2000;
        ALLOCATE STMT(0:MAXSTNO);

/*----------------------------------------------------------*/
/* ENDPAGE-CONDITION-ON-UNIT FOR THE DEFINITION OF A NEW    */
/* PAGE AND FOR THE OUTPUT OF THE NUMBER OF THE PAGE:       */
/*----------------------------------------------------------*/

ON      ENDPAGE(SYSPRINT) BEGIN;
        PAGENO  = PAGENO+1;
        PUT PAGE EDIT('PAGE ',PAGENO) (COL(62),A,F(4));
        PUT SKIP(2);
END;

/*----------------------------------------------------------*/
/* THE FOLLOWING ON-CONDITIONS DETECT EACH ERROR CONCERN-   */
/* ING EACH OF THE FILES USED IN PRISM (EXCEPT SYSPRINT),   */
/* AS FAR AS THE ERROR IS NOT RECOGNIZED BY THE PROCE-      */
/* DURES ACTIVATED BY MONITOR. IF AN ERROR OCCURS, THE      */
/* ON-CONDITION-ON-UNITS ACTIVATE THE INTERNAL PROCEDURE    */
/* FLE_ERR, WHICH WRITES AN APPROPRIATE ERROR-MESSAGE ON-   */
/* TO THE SYSPRINT-FILE:                                    */
/*----------------------------------------------------------*/

ON      UNDEFINEDFILE(CORPUS)   CALL FLE_ERR('CORPUS');
ON      RECORD        (CORPUS)  CALL FLE_ERR('CORPUS');
ON      TRANSMIT      (CORPUS)  CALL FLE_ERR('CORPUS');
ON      UNDEFINEDFILE(STMTFLE)  CALL FLE_ERR('STMTFLE');
ON      RECORD        (STMTFLE) CALL FLE_ERR('STMTFLE');
ON      TRANSMIT      (STMTFLE) CALL FLE_ERR('STMTFLE');
ON      UNDEFINEDFILE(SYSIN)    CALL FLE_ERR('SYSIN');
ON      TRANSMIT      (SYSIN)   CALL FLE_ERR('SYSIN');
ON      UNDEFINEDFILE(WRK1FLE)  CALL FLE_ERR('WRK1FLE');
ON      RECORD        (WRK1FLE) CALL FLE_ERR('WRK1FLE');
ON      TRANSMIT      (WRK1FLE) CALL FLE_ERR('WRK1FLE');
ON      UNDEFINEDFILE(WRK2FLE)  CALL FLE_ERR('WRK2FLE');
ON      RECORD        (WRK2FLE) CALL FLE_ERR('WRK2FLE');
ON      TRANSMIT      (WRK2FLE) CALL FLE_ERR('WRK2FLE');
ON      UNDEFINEDFILE(WRK3FLE)  CALL FLE_ERR('WRK3FLE');
ON      KEY           (WRK3FLE) CALL FLE_ERR('WRK3FLE');
ON      TRANSMIT      (WRK3FLE) CALL FLE_ERR('WRK3FLE');
```

```
FLE_ERR: PROC(FILENAME);
DCL      FILENAME              CHAR(7) VAR;

         SIGNAL ENDPAGE(SYSPRINT);
         PUT SKIP EDIT(
         '*** ERROR WHILE TRYING TO ACCESS THE DATA SET AL',
         'LOCATED TO',
         '    FILE '||FILENAME||' AND BREAK OF PRISM. ',
         'PLEASE CHECK IF YOU',
         '    HAVE ALLOCATED A DATA SET TO '||FILENAME||'.',
         ' MAKE SURE THAT',
         '    THIS DATA SET HAS APPROPRIATE ATTRIBUTES AND',
         ' IS ALLOCAT-',
         '    ED WITH SUFFICIENT SPACE.')
         (COL(11),2 A);
         GOTO EXIT;

END      FLE_ERR;

/*-------------------------------------------------------------*/
/* OUTPUT OF THE FRONT PAGE ONTO THE SYSPRINT-FILE:            */
/*-------------------------------------------------------------*/

         STRING(DAT) = DATE;
         STRING(TIM) = TIME;
         PAGENO      = 1;
         PUT EDIT
         ('DATE: ',DAT(3),'.',DAT(2),'.',DAT(1),'TIME: ',
         TIM(1),':',TIM(2),':',TIM(3),'PAGE ',PAGENO)
         (COL(11),6 A,X(4),6 A,COL(62),A,F(4));
         PUT SKIP(4);
         CALL PTPRISM;
```

```
PTPRISM: PROC;

         PUT EDIT(
'PPPPPPPP   RRRRRRRR   IIIIIIIII   SSSSSSS   M        M',
'PPPPPPPPP  RRRRRRRRR  IIIIIIIII  SSSSSSSSS  MM      MM',
'PP     PP  RR      RR     II     SS      SS  MMM    MMM',
'PP     PP  RR      RR     II     SS          MMMM  MMMM',
'PP     PP  RR      RR     II     SSS         MM MMMM MM',
'PPPPPPPPP  RRRRRRRRR      II      SSSSSS     MM  MM  MM',
'PPPPPPPP   RRRPRRRR       II       SSSSSS    MM      MM',
'PP         RR  RR         II            SSS  MM      MM',
'PP         RR   RR        II             SS  MM      MM',
'PP         RR    RR       II      SS     SS  MM      MM',
'PP         RP     RR  IIIIIIIII  SSSSSSSSS  MM      MM',
'PP         RR      RR IIIIIIIII   SSSSSSS   MM      MM')
         (COL(12),A);

END      PTPRISM;

         PUT SKIP(4) EDIT(
  'V   V  EEEEE  RRRR   SSSSS  IIIII  OOOOO  N   N      11',
  'V   V  E      R   R  S        I    O   O  NN  N     1 11',
  'V   V  EEE    RRRR   SSSSS    I    O   O  N N N   1   1',
  ' V V   E      R R        S   I    O   O  N  NN       1',
  '  V    EEEEE  R  R   SSSSS  IIIII  OOOOO  N   N      1')
         (COL(13),A);
         PUT LINE(46) EDIT
         ('AUTHOR: K. WOTHKE')              (COL(12),A)
         ('INSTITUT FUER KOMMUNIKATIONS-')  (COL(20),A)
         ('FORSCHUNG UND PHONETIK')         (COL(20),A)
         ('UNIVERSITY OF BONN')             (COL(20),A)
         ('POPPELSDORFER ALLEE 47')         (COL(20),A)
         ('53 BONN 1')              (SKIP(2),COL(20),A)
         ('GERMANY (WEST)')         (SKIP(2),COL(20),A);

/*-----------------------------------------------------------*/
/* INPUT OF THE OPTIONS:                                     */
/*-----------------------------------------------------------*/

         CALL GETOPTN(MODE,PRINT,STATISTICS,RETURN_CODE);
            IF RETURN_CODE>0 THEN GOTO EXIT;
```

```
/*-------------------------------------------------------*/
/* ACTIVATION OF THE ALGORITHM INDICATED BY THE VALUE OF  */
/* MODE:                                                  */
/*-------------------------------------------------------*/

SELECT(MODE);

        /*-----------------------------------------------*/
        /* ACTIVATION OF THE LEARNING-ALGORITHM:         */
        /*-----------------------------------------------*/

 WHEN('LEARN') DO;
        CALL CHKCRPS(RETURN_CODE);
           IF RETURN_CODE=0 THEN
        CALL DISCOV(RETURN_CODE);
           IF RETURN_CODE>0 THEN DO;
            OPEN  FILE(STMTFLE) OUTPUT;
            CLOSE FILE(STMTFLE);
            GOTO EXIT;
            END;
           IF PRINT THEN
        CALL STMTOUT(STATISTICS);
        CALL TODSET;
 END;

        /*-----------------------------------------------*/
        /* ACTIVATION OF THE ALGORITHM FOR THE APPLICA-  */
        /* TION OF DERIVATION-STATEMENTS:                */
        /*-----------------------------------------------*/

 WHEN('APPLY') DO;
        CALL FRODSET('APPLY',RETURN_CODE);
           IF RETURN_CODE>0 THEN GOTO EXIT;
        CALL APPLY;
           IF PRINT THEN
        CALL LISTOUT;
 END;

        /*-----------------------------------------------*/
        /* ACTIVATION OF THE ALGORITHM FOR THE OUTPUT OF */
        /* DERIVATION-STATEMENTS, WHICH WERE INFERED DU- */
        /* RING A FORMER ACTIVATION OF THE LEARNING-AL-  */
        /* GORITHM:                                      */
        /*-----------------------------------------------*/

 WHEN('PRINT_STMTS') DO;
        CALL FRODSET('PRINT_STMTS',RETURN_CODE);
           IF RETURN_CODE>0 THEN GOTO EXIT;
        CALL STMTOUT(STATISTICS);
 END;
END;
```

```
/*-------------------------------------------------------*/
/* FREEING OF THE STORAGE ASSOCIATED WITH THE STRUCTURE- */
/* ARRAY STMT AND OUTPUT OF THE LAST PAGE ONTC THE       */
/* SYSPRINT-FILE:                                        */
/*-------------------------------------------------------*/

EXIT:    FREE STMT;
         SIGNAL ENDPAGE(SYSPRINT);
         PUT SKIP EDIT(
         'EEEEE  N   N  DDDD          OOOOO  FFFFF',
         'E      NN  N  D   D         O   O  F    ',
         'EEE    N N N  D   D         O   O  FFF  ',
         'E      N  NN  D   D         O   O  F    ',
         'EEEEE  N   N  DDDD          OOOOO  F    ')
         (COL(20),A);
         PUT SKIP(4);
         CALL PTPRISM;

END      MONITOR;
```

```
/*-------------------------------------------------------------*/
/* NAME: GETOPTN                                               */
/*                                                             */
/* FUNCTION:                                                   */
/* GETOPTN IS A SUBROUTINE-PROCEDURE. IT READS THE OP-         */
/* TION-FIELD, WHICH CONTAINS THE OPTIONS NECESSARY FOR        */
/* THE CONTROL OF PRISM, FROM THE SYSIN-FILE. THE OPTION-      */
/* FIELD IS THEN CHECKED FOR FORMAL ERRORS, AND ALL FOR-       */
/* MAL ERRORS ARE DETECTED. IF THERE ARE ERRORS, GETOPTN       */
/* WRITES FOR EACH ERROR AN ERROR-MESSAGE ONTO THE             */
/* SYSPRINT-FILE, AND IT ASSIGNS TO ITS PARAMETER              */
/* RETURN_CODE THE VALUE 1. IF THERE ARE NOT ANY ERRORS,       */
/* GETOPTN ASSIGNS THE SPECIFIED OPTIONS TO ITS PARAME-        */
/* TERS MODE, PRINT AND STATISTICS, AND IT WRITES THE          */
/* SPECIFIED AND THE USED OPTIONS ONTO THE SYSPRINT-FILE.      */
/*                                                             */
/* PARAMETERS:                                                 */
/*  MODE:          CHAR(11) VAR. IF THERE ARE NOT ANY ER-      */
/*                 RORS IN THE OPTION-FIELD, GETOPTN AS-       */
/*                 SIGNS TO MODE A STRING WHICH INDICATES      */
/*                 THE COMPONENT OF PRISM WHICH IS TO BE       */
/*                 ACTIVATED. 3 DIFFERENT STRINGS ARE POS-     */
/*                 SIBLE, WHICH CORRESPOND TO THE 3 COMPO-     */
/*                 NENTS PRISM:                                */
/*                  'LEARN'        - THE LEARNING-ALGORITHM    */
/*                                   SHALL BE ACTIVATED.       */
/*                  'APPLY'        - THE ALGORITHM FOR THE     */
/*                                   APPLICATION OF DERIVA-    */
/*                                   TION-STATEMENTS SHALL     */
/*                                   BE ACTIVATED.             */
/*                  'PRINT_STMTS'  - THE ALGORITHM FOR THE     */
/*                                   OUTPUT OF DERIVATION-     */
/*                                   STATEMENTS, WHICH WERE    */
/*                                   INFERED DURING A FORMER   */
/*                                   ACTIVATION OF THE         */
/*                                   LEARNING-ALGORITHM, ON-   */
/*                                   TO THE SYSPRINT-FILE      */
/*                                   SHALL BE ACTIVATED.       */
/*  PRINT:         BIT(1). IF GETOPTN ASSIGNS TO MODE THE      */
/*                 STRING 'LEARN' OR THE STRING 'APPLY', IT    */
/*                 ALSO ASSIGNS A BIT-STRING TO PRINT. THIS    */
/*                 BIT-STRING INDICATES WHETHER THE RESULT     */
/*                 OF THE OPERATION INDICATED BY THE VALUE     */
/*                 OF MODE IS TO BE WRITTEN ONTO THE           */
/*                 SYSPRINT-FILE:                              */
/*                  '0'B - THE RESULT SHALL NOT BE WRITTEN     */
/*                         ONTO THE SYSPRINT-FILE,             */
/*                  '1'B - THE RESULT IS TO BE WRITTEN ONTO    */
/*                         THE SYSPRINT-FILE.                  */
/*  STATISTICS:    BIT(1). IF GETOPTN ASSIGNS TO MODE AND/     */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------*/
/*               OR PRINT VALUES WHICH INDICATE THAT A         */
/*               SET OF DERIVATION-STATEMENTS IS TO BE         */
/*               WRITTEN ONTO THE SYSPRINT-FILE, IT ALSO       */
/*               ASSIGNS A BIT-STRING TO STATISTICS. THIS      */
/*               BIT-STRING INDICATES WHETHER THE DERIVA-      */
/*               TION-STATEMENTS ARE TO BE PRINTED WITH        */
/*               OR WITHOUT STATISTICAL COMPUTATIONS:          */
/*                '0'B - NO STATISTICAL COMPUTATIONS ARE       */
/*                       TO BE PRINTED,                        */
/*                '1'B - STATISTICAL COMPUTATIONS ARE TO       */
/*                       BE PRINTED.                           */
/*  RETURN_CODE: BIN FIXED(31). IF THERE ARE FORMAL ER-        */
/*               RORS IN THE OPTION-FIELD, GETOPTN AS-         */
/*               SIGNS TO RETURN_CODE THE VALUE 1.             */
/*                                                             */
/* FILES:                                                      */
/*  INPUT:  SYSIN,                                             */
/*  OUTPUT: SYSPRINT.                                          */
/*-------------------------------------------------------------*/

GETOPTN: PROC(MODE,PRINT,STATISTICS,RETURN_CODE) REORDER;
DCL      MODE                    CHAR(11) VAR,
         OPTION_FIELD            CHAR(50),
         CUT_OPTION_FIELD(50)    CHAR(1) DEF OPTION_FIELD
                                    POS(1),
         POT_OPT(25)             CHAR(52) VAR,
         CONC_POT_OPTS           CHAR(1300) VAR INIT(''),
         BIT_REP                 BIT(8),
         PRINT                   BIT(1),
         STATISTICS              BIT(1),
         NO_OF_OPTS              BIN FIXED(15) INIT(0),
         (I,J)                   BIN FIXED(15),
         RETURN_CODE             BIN FIXED(31),
         (COUNT,INDEX,LENGTH)    BUILTIN,
         (SUBSTR,UNSPEC)         BUILTIN,
         SYSIN                   FILE STREAM INPUT,
         SYSPRINT                FILE STREAM PRINT;

/*-------------------------------------------------------------*/
/* OUTPUT OF THE OPTION-FIELD IF SYSIN IS EMPTY:               */
/*-------------------------------------------------------------*/

ON       ENDFILE(SYSIN) BEGIN;
         SIGNAL ENDPAGE(SYSPRINT);
```

```
         PUT SKIP EDIT
         ('GIVEN OPTION-FIELD:')  (COL(11),A)
         ((19)'-')                (COL(11),A)
         ('''''')                 (SKIP(2),COL(11),A)
         ('ERROR-MESSAGES FOR',
         ' THE OPTION-FIELD:')    (SKIP(3),COL(11),2 A)
         ((36)'-')                (COL(11),A);
         OPTION_FIELD = '';
         GOTO CONT;
END;

/*-------------------------------------------------------------*/
/* INPUT OF THE OPTION-FIELD FROM SYSIN:                       */
/*-------------------------------------------------------------*/

         GET EDIT(OPTION_FIELD)  (A(50));

/*-------------------------------------------------------------*/
/* CONVERSION OF LOWER CASE LETTERS TO CAPITAL LETTERS         */
/* (THE USE OF THE EBCDI-CODE IS PRESUPPOSED):                 */
/*-------------------------------------------------------------*/

DO       I=1 TO 50;
         BIT_REP = UNSPEC(OUT_OPTION_FIELD(I));
         IF '10000001'B<=BIT_REP & BIT_REP<='10001001'B
          | '10010001'B<=BIT_REP & BIT_REP<='10011001'B
          | '10100010'B<=BIT_REP & BIT_REP<='10101001'B
          THEN UNSPEC(OUT_OPTION_FIELD(I))=BIT_REP | '01'B;
END;

/*-------------------------------------------------------------*/
/* OUTPUT OF THE OPTION-FIELD ONTO THE SYSPRINT-FILE:          */
/*-------------------------------------------------------------*/

         SIGNAL ENDPAGE(SYSPRINT);
         PUT SKIP EDIT
         ('GIVEN OPTION-FIELD:')   (COL(11),A)
         ((19)'-')                 (COL(11),A)
         ('''',OPTION_FIELD,'''')  (SKIP(2),COL(11),3 A);

/*-------------------------------------------------------------*/
/* SEGMENTATION OF THE OPTION-FIELD INTO POTENTIAL OP-         */
/* TION-KEYWORDS AND STORAGE OF THESE ISOLATED KEYWORDS        */
/* IN THE ARRAY POT_OPT AND - NEWLY CONCATENATED IN A          */
/* SPECIAL FORMAT - IN THE VARIABLE CONC_POT_OPTS:             */
/*-------------------------------------------------------------*/

DO       I=1 TO 50;
```

```
        IF CUT_OPTION_FIELD(I)¬=' '
 THEN DO;
  DO    J=I+1 TO 50 UNTIL(CUT_OPTION_FIELD(J)=' ');
  END;
        NO_OF_OPTS          = NO_OF_OPTS+1;
        POT_OPT(NO_OF_OPTS) = ' '||SUBSTR(OPTION_FIELD,I,
                              J-I)||' ';
        CONC_POT_OPTS       = CONC_POT_OPTS||
                              POT_OPT(NO_OF_OPTS);
        I                   = J;
 END;
END;

/*-------------------------------------------------------*/
/* CHECKING OF THE OPTION-FIELD FOR SYNTACTIC-ERRORS     */
/* (BASED ON THE ANALYSIS OF THE DATA IN POT_OPT AND IN  */
/* CONC_POT_OPTS) AND - FOR EACH ERROR - OUTPUT OF AN ER- */
/* ROR-MESSAGE ONTO THE SYSPRINT-FILE:                   */
/*-------------------------------------------------------*/

        PUT SKIP(3) EDIT
        ('ERROR-MESSAGES FOR THE OPTION-FIELD:',(36)'-')
        (COL(11),A);
        PUT;

        /*-----------------------------------------------*/
        /* CHECK OF EACH POTENTIAL OPTION-KEYWORD IN THE */
        /* OPTION-FIELD WHETHER IT IS AN ADMISSIBLE OP-  */
        /* TION-KEYWORD:                                 */
        /*-----------------------------------------------*/

DO      I=1 TO NO_OF_OPTS;
        IF INDEX(' LEARN APPLY PRINT_STMTS PRINT NOPRINT'||
                 ' STATISTICS NOSTATISTICS ',POT_OPT(I))=0
 THEN DO;
        CALL LINEFD(57);
        PUT SKIP EDIT
        ('*** STRING SPECIFIED IS NOT AN OPTION-KEYWORD:',
        '    ''',SUBSTR(POT_OPT(I),2,LENGTH(POT_OPT(I))-2),
        ''''.')
        (COL(11),A,COL(11),3 A);
 END;
END;
```

```
        /*-----------------------------------------------*/
        /* CHECK OF THE OPTION-FIELD ON THE CONSISTENCY  */
        /* OF THE  E X P L I C I T L Y  SPECIFIED OP-    */
        /* TIONS:                                        */
        /*-----------------------------------------------*/

        CALL CHKCONS('LEARN','APPLY');
        CALL CHKCONS('LEARN','PRINT_STMTS');
        CALL CHKCONS('APPLY','PRINT_STMTS');
        CALL CHKCONS('PRINT','NOPRINT');
        CALL CHKCONS('STATISTICS','NOSTATISTICS');
        CALL CHKCONS('APPLY','STATISTICS');
        CALL CHKCONS('APPLY','NOSTATISTICS');
        CALL CHKCONS('PRINT_STMTS','PRINT');
        CALL CHKCONS('PRINT_STMTS','NOPRINT');
        CALL CHKCONS('NOPRINT','STATISTICS');
        CALL CHKCONS('NOPRINT','NOSTATISTICS');

CHKCONS: PROC(KEYWORD_1,KEYWORD_2);
DCL     KEYWORD_1           CHAR(12) VAR,
        KEYWORD_2           CHAR(12) VAR;

        IF INDEX(CONC_POT_OPTS,' '||KEYWORD_1||' ')>0
         & INDEX(CONC_POT_OPTS,' '||KEYWORD_2||' ')>0
THEN DO;
        CALL LINEFD(57);
        PUT SKIP EDIT
        ('*** INCONSISTENT OPTIONS. ',
        'THE 2 EXCLUSIVE OPTIONS')       (COL(11),2 A)
        ('    ''',KEYWORD_1,''' AND ''',
        KEYWORD_2,''' WERE SPECIFIED.')  (COL(11),5 A);
END;

END     CHKCONS;

        /*-----------------------------------------------*/
        /* CHECK OF THE OPTION-FIELD ON THE CONSISTENCY  */
        /* OF THE  E X P L I C I T L Y  AND THE  I M -   */
        /* P L I C I T L Y  SPECIFIED OPTIONS:           */
        /*-----------------------------------------------*/

        IF INDEX(CONC_POT_OPTS,' LEARN ')=0
         & INDEX(CONC_POT_OPTS,' APPLY ')=0
         & INDEX(CONC_POT_OPTS,' PRINT_STMTS ')=0
THEN DO;
        IF INDEX(CONC_POT_OPTS,' STATISTICS ')>0
 THEN DO;
        CALL LINEFD(57);
```

```
         PUT SKIP EDIT
         ('*** THE ASSUMED MODE ''APPLY''',
         ' IS INCONSISTENT WITH THE SPEC-')     (COL(11),2 A)
         ('    IFICATION OF ''STATISTICS''.')   (COL(11),A);
 END;
         IF INDEX(CONC_POT_OPTS,' NOSTATISTICS ')>0
 THEN DO;
         CALL LINEFD(57);
         PUT SKIP EDIT
         ('*** THE ASSUMED MODE ''APPLY''',
         ' IS INCONSISTENT WITH THE SPEC-')     (COL(11),2 A)
         ('    IFICATION OF ''NOSTATISTICS''.') (COL(11),A);
 END;
END;

         /*-----------------------------------------------*/
         /* CHECK OF THE OPTION-FIELD ON THE MULTIPLE     */
         /* SPECIFICATION OF IDENTICAL OPTION-KEYWORDS:   */
         /*-----------------------------------------------*/

         CALL CHKMSPC('LEARN');
         CALL CHKMSPC('APPLY');
         CALL CHKMSPC('PRINT_STMTS');
         CALL CHKMSPC('PRINT');
         CALL CHKMSPC('NOPRINT');
         CALL CHKMSPC('STATISTICS');
         CALL CHKMSPC('NOSTATISTICS');

CHKMSPC: PROC(KEYWORD);
DCL      KEYWORD              CHAR(12) VAR;

         I = INDEX(CONC_POT_OPTS,' '||KEYWORD||' ');
         IF I>0
          & INDEX(SUBSTR(CONC_POT_OPTS,I+LENGTH(KEYWORD)+2),
            ' '||KEYWORD||' ')>0
THEN DO;
         CALL LINEFD(58);
         PUT SKIP EDIT
         ('*** THE OPTION ''',KEYWORD,
         ''' WAS REPEATEDLY SPECIFIED.')  (COL(11),3 A);
END;

END      CHKMSPC;
```

```
        /*-------------------------------------------------*/
        /* OUTPUT OF A SUMMARIZING MESSAGE CONCERNING      */
        /* THE ERROR-MESSAGES:                             */
        /*-------------------------------------------------*/

        IF COUNT(SYSPRINT)>0
THEN DO;
        RETURN_CODE = 1;
        PUT SKIP(2) EDIT
        ('BECAUSE OF THE ERROR(S)',
        ' PRISM CANNOT BE EXECUTED.')   (COL(11),2 A);
END;
ELSE DO;
CONT:   PUT SKIP(2) EDIT
        ('THE OPTION-FIELD DOES NOT',
        ' CONTAIN ANY FORMAL ERRORS.')  (COL(11),2 A);

/*-------------------------------------------------------*/
/* EXECUTION OF THE NECESSARY OPERATIONS FOR AN OPTION-  */
/* FIELD WITHOUT ERRORS:                                 */
/*-------------------------------------------------------*/
        /*-------------------------------------------------*/
        /* OUTPUT OF THE SPECIFIED OPTIONS:                */
        /*-------------------------------------------------*/

        PUT SKIP(3) EDIT
        ('OPTIONS SPECIFIED:')  (COL(11),A)
        ((18)'-')               (COL(11),A);
        PUT SKIP EDIT
        ((SUBSTR(POT_OPT(I),2)
        DO I=1 TO NO_OF_OPTS))  (SKIP,COL(11),A);

        /*-------------------------------------------------*/
        /* ASSIGNMENT OF THE USED OPTIONS TO THE OPTION-   */
        /* VARIABLES AND OUTPUT OF THE USED OPTIONS:       */
        /*-------------------------------------------------*/

        PUT SKIP(3) EDIT
        ('OPTIONS USED:',(13)'-')  (COL(11),A);
        PUT SKIP(2);
 SELECT;
  WHEN(INDEX(OPTION_FIELD,'LEARN')>0) DO;
        IF INDEX(OPTION_FIELD,'NOPRINT')>0
   THEN DO;
        MODE        = 'LEARN';
        PRINT       = '0'B;
        PUT EDIT('LEARN','NOPRINT')            (COL(11),A);
```

```
    END;
    ELSE DO;
         IF INDEX(OPTION_FIELD,'NOSTATISTICS')>0
     THEN DO;
         MODE       = 'LEARN';
         PRINT      = '1'B;
         STATISTICS = '0'B;
         PUT EDIT
              ('LEARN','PRINT','NOSTATISTICS') (COL(11),A);
     END;
     ELSE DO;
         MODE       = 'LEARN';
         PRINT      = '1'B;
         STATISTICS = '1'B;
         PUT EDIT('LEARN','PRINT','STATISTICS') (COL(11),A);
     END;
    END;
   END;
   WHEN(INDEX(OPTION_FIELD,'PRINT_STMTS')>0) DO;
         IF INDEX(OPTION_FIELD,'NOSTATISTICS')>0
    THEN DO;
         MODE       = 'PRINT_STMTS';
         STATISTICS = '0'B;
         PUT EDIT('PRINT_STMTS','NOSTATISTICS') (COL(11),A);
    END;
    ELSE DO;
         MODE       = 'PRINT_STMTS';
         STATISTICS = '1'B;
         PUT EDIT('PRINT_STMTS','STATISTICS')   (COL(11),A);
    END;
   END;
   OTHERWISE DO;
         IF INDEX(' '||OPTION_FIELD,' PRINT')>0
    THEN DO;
         MODE       = 'APPLY';
         PRINT      = '1'B;
         PUT EDIT('APPLY','PRINT')              (COL(11),A);
    END;
    ELSE DO;
         MODE       = 'APPLY';
         PRINT      = '0'B;
         PUT EDIT('APPLY','NOPRINT')            (COL(11),A);
    END;
   END;
  END;
 END;
```

```
/*-----------------------------------------------------------*/
/* SUBROUTINE-PROCEDURE FOR THE SKIPPING OF A LINE OR FOR */
/* THE DEFINITION OF A NEW PAGE BEFORE A MESSAGE IS WRIT- */
/* TEN ONTO THE SYSPRINT-FILE:                               */
/*-----------------------------------------------------------*/

LINEFD:  PROC(MAX_LINENO);
DCL      MAX_LINENO          BIN FIXED(15),
         LINENO              BUILTIN;

         IF LINENO(SYSPRINT)<=MAX_LINENO
          THEN PUT SKIP;
          ELSE SIGNAL ENDPAGE(SYSPRINT);

END      LINEFD;

END      GETOPTN;
```

```
/*-------------------------------------------------------*/
/* NAME: CHKCRPS                                         */
/*                                                       */
/* FUNCTION:                                             */
/* CHKCRPS IS A SUBROUTINE-PROCEDURE. IT DETECTS ALL FOR- */
/* MAL ERRORS IN A LEARNING-CORPUS. IF THERE ARE FORMAL   */
/* ERRORS, CHKCRPS WRITES FOR EACH ERROR AN ERROR-MESSAGE */
/* ONTO THE SYSPRINT-FILE, AND IT ASSIGNS THE VALUE 1 TO  */
/* ITS PARAMETER RETURN_CODE. TO DETECT ALL ERRORS IN THE */
/* LEARNING-CORPUS, CHKCRPS MUST ACTIVATE THE SORT-UTILI- */
/* TY SYNCSORT. IF THE EXECUTION OF SYNCSORT FAILS, THE   */
/* VALUE 16 IS ASSIGNED TO RETURN_CODE.                   */
/*    FURTHERMORE CHKCRPS COMPUTES THE NUMBER OF SOURCE-  */
/* TARGET-PAIRS IN A LEARNING-CORPUS (= RECNO_CORPUS).    */
/*                                                       */
/* PARAMETER:                                            */
/*  RETURN_CODE: BIN FIXED(31). IF AN ERROR OCCURS DURING */
/*               THE EXECUTION OF CHKCRPS, CHKCRPS AS-   */
/*               SIGNS TO RETURN_CODE AN INTEGER GREATER */
/*               THAN 0. IN DETAIL, THE ASSIGNED INTEGER */
/*               MEANS:                                  */
/*                 1 - THERE ARE FORMAL ERRORS IN THE    */
/*                     LEARNING-CORPUS,                  */
/*                16 - THE EXECUTION OF THE SORT-UTILITY */
/*                     SYNCSORT, WHICH IS CALLED BY      */
/*                     CHKCRPS VIA THE INTERFACE PROGRAM */
/*                     PLISRTA, FAILED.                  */
/*                                                       */
/* FILES:                                                */
/*  INPUT:        CORPUS,                                */
/*  INPUT/OUTPUT: WRK1FLE,                               */
/*  OUTPUT:       SYSPRINT.                              */
/*    FURTHERMORE THE SORT-UTILITY SYNCSORT, WHICH IS    */
/* CALLED BY CHKCRPS VIA THE INTERFACE PROGRAM PLISRTA,   */
/* NEEDS THE FILES: WRK1IN, WRK1OUT, WRK1WK01, WRK1WK02   */
/* AND WRK1WK03.                                         */
/*-------------------------------------------------------*/

CHKCRPS: PROC(RETURN_CODE) REORDER;
DCL      1 BOOKEEP              EXT,
          2 RECNO_CORPUS        BIN FIXED(31) INIT(0),
          2 LST_PRF_BLCK_STMT   BIN FIXED(15),
          2 LST_SUF_BLCK_STMT   BIN FIXED(15),
          2 LST_PRF_SBST_STMT   BIN FIXED(15),
          2 LST_SUF_SBST_STMT   BIN FIXED(15),
          2 DUMMY               CHAR(58),

         CRC_1                  CHAR(80),
          DUMMY_1               CHAR(10) DEF CRC_1 POS(1),
          CRC_1_SOURCE          CHAR(30) DEF CRC_1 POS(11),
          TARGET                CHAR(30) DEF CRC_1 POS(41),
          DUMMY_2               CHAR(10) DEF CRC_1 POS(71),
```

```
        1 CRC_2,
          2 RECPNT_CORPUS_BI   BIN FIXED(31),
          2 SOURCE             CHAR(30),
          2 DUMMY              CHAR(39) INIT(' '),

        LST_SRC                CHAR(30),
        WRKCHR                 CHAR(60) VAR,
        RECPNT_CORPUS_CH       PIC 'ZZZZZ9' INIT(0),
        ERR                    BIT(2) INIT('00'B),
        SRC_ERR                BIT(1) DEF ERR POS(1),
        TRG_ERR                BIT(1) DEF ERR POS(2),
        LST_RECPNT             BIN FIXED(31),
        RETURN_CODE            BIN FIXED(31),
        (COUNT,FLOOR,HIGH)     BUILTIN,
        (INDEX,LENGTH,LOW)     BUILTIN,
        (PLISRTA,SUBSTR)       BUILTIN,
        VERIFY                 BUILTIN,
        CORPUS                 FILE RECORD SEQI,
        WRK1FLE                FILE RECORD SEQI,
        SYSPRINT               FILE STREAM PRINT;

/*-------------------------------------------------------*/
/* OUTPUT OF A HEADING:                                  */
/*-------------------------------------------------------*/

        SIGNAL ENDPAGE(SYSPRINT);
        PUT SKIP EDIT
        ('ERROR-MESSAGES FOR THE LEARNING-CORPUS:',
        (39)'-') (COL(11),A);
        PUT;

/*-------------------------------------------------------*/
/*  - COMPUTATION OF THE NUMBER OF SOURCE-TARGET-PAIRS   */
/*    (= RECNO_CORPUS) IN THE LEARNING-CORPUS,           */
/*  - SYNTAX-CHECKING OF THE LEARNING-CORPUS (EXCEPT FOR */
/*    THE MULTIPLE OCCURRENCE OF SOURCES) AND FOR EACH   */
/*    ERROR OUTPUT OF AN ERROR-MESSAGE,                  */
/*  - COPYING OF ALL SOURCE-TARGET-PAIRS WHICH DO NOT    */
/*    CONTAIN AN ERROR IN THE SOURCE-FIELD FROM CORPUS TO */
/*    WRK1FLE:                                           */
/*-------------------------------------------------------*/

ON      ENDFILE(CORPUS) GOTO SHUT_1;
        OPEN FILE(CORPUS)  INPUT,
             FILE(WRK1FLE) OUTPUT;
DO      WHILE('1'B);
        READ FILE(CORPUS) INTO(CRC_1);
```

```
          RECNO_CORPUS      = RECNO_CORPUS+1;

          IF INDEX(CRC_1_SOURCE,' ')=1
           | VERIFY(SUBSTR(CRC_1_SOURCE||' ',
             INDEX(CRC_1_SOURCE||' ',' ')),' ')>0
           | INDEX(CRC_1_SOURCE,LOW(1))>0
           | INDEX(CRC_1_SOURCE,HIGH(1))>0
 THEN      SRC_ERR           = '1'B;
 ELSE DO;
          CRC_2.SOURCE      = CRC_1_SOURCE;
          RECPNT_CORPUS_BI = RECNO_CORPUS;
          WRITE FILE(WRK1FLE) FROM(CRC_2);
 END;
          IF VERIFY(SUBSTR(TARGET||' ',
             INDEX(TARGET||' ',' ')),' ')>0
           | INDEX(TARGET,LOW(1))>0
           | INDEX(TARGET,HIGH(1))>0
           THEN TRG_ERR      = '1'B;

          IF ERR
 THEN DO;
          CALL LINEFD(57-FLOOR(ERR/3));
          RECPNT_CORPUS_CH = RECNO_CORPUS;
          PUT SKIP EDIT
          ('*** LINE ',SUBSTR(RECPNT_CORPUS_CH,
          VERIFY(RECPNT_CORPUS_CH,' ')),':')
          (COL(11),3 A);
          IF SRC_ERR
           THEN PUT SKIP EDIT
                ('    ERROR IN SOURCE-FIELD: ''',
               CRC_1_SOURCE,'''.') (COL(11),3 A);
          IF TRG_ERR
           THEN PUT SKIP EDIT
                ('    ERROR IN TARGET-FIELD: ''',
               TARGET,'''.')        (COL(11),3 A);
          ERR               = '00'B;
 END;
END;
SHUT_1:   CRC_2.SOURCE      = HIGH(1);
          WRITE FILE(WRK1FLE) FROM(CRC_2);
          CLOSE FILE(CORPUS),
                FILE(WRK1FLE);
```

```
/*-------------------------------------------------------*/
/* SORTING OF WRK1FLE ON THE SOURCES AND ON THE LINE-NUM- */
/* BERS IN ASCENDING ORDER AND - EVENTUALLY - OUTPUT OF   */
/* AN ERROR-MESSAGE:                                      */
/*-------------------------------------------------------*/

          CALL PLISRTA(' SORT FIELDS=(5,30,CH,A,1,4,FI,A) ',
                       ' RECORD TYPE=F,LENGTH=73 ',
                       100000,RETURN_CODE,'WRK1');
          IF RETURN_CODE¬=0
THEN DO;
          SIGNAL ENDPAGE(SYSPRINT);
          PUT SKIP EDIT(
          '*** ERROR DURING SORTING OF WRK1FLE AND BREAK OF',
          ' PRISM.',
          '    PLEASE CHECK THE ALLOCATION OF DATA SETS TO ',
          'THE FILES',
          '    WRK1IN, WRK1OUT, WRK1WK01, WRK1WK02 AND ',
          'WRK1WK03. MAKE',
          '    SURE THAT THESE DATA SETS HAVE APPROPRIATE ',
          'ATTRIBUTES',
          '    AND ARE ALLOCATED WITH SUFFICIENT SPACE.')
          (COL(11),2 A);
          RETURN;
END;

/*-------------------------------------------------------*/
/* CHECKING OF THE LEARNING-CORPUS FOR THE MULTIPLE OC-   */
/* CURRENCE OF SOURCES (BASED ON THE ANALYSIS OF THE      */
/* SORTED SOURCES IN WRK1FLE) AND FOR EACH MULTIPLE OC-   */
/* CURRENCE OUTPUT OF AN ERROR-MESSAGE:                   */
/*-------------------------------------------------------*/

ON        ENDFILE(WRK1FLE) GOTO SHUT_2;
          OPEN FILE(WRK1FLE) INPUT;
          LST_SRC          = LOW(1);
DO        WHILE('1'B);
          READ FILE(WRK1FLE) INTO(CRC_2);
          IF CRC_2.SOURCE=LST_SRC THEN
 DO;
          CALL LINEPD(57);
          RECPNT_CORPUS_CH = LST_RECPNT;
          WRKCHR = '*** LINES '||SUBSTR(RECPNT_CORPUS_CH,
                   VERIFY(RECPNT_CORPUS_CH,' ')),||', ';
  DO      UNTIL(CRC_2.SOURCE¬=LST_SRC);
          RECPNT_CORPUS_CH = RECPNT_CORPUS_BI;
```

```
        IF LENGTH(WRKCHR)+9-VERIFY(RECPNT_CORPUS_CH,' ')>60
   THEN DO;
        PUT SKIP EDIT(WRKCHR) (COL(11),A);
        WRKCHR          = (10)' ';
   END;
        WRKCHR          = WRKCHR||SUBSTR(RECPNT_CORPUS_CH,
                          VERIFY(RECPNT_CORPUS_CH,' '))||
                          ', ';
        READ FILE(WRK1FLE) INTO(CRC_2);
  END;
        SUBSTR(WRKCHR,LENGTH(WRKCHR)-1)=':';
        PUT SKIP EDIT
        (WRKCHR)                                (COL(11),A)
        ('    MULTIPLE SOURCE: ''',SUBSTR(
        LST_SRC||' ',1,INDEX(LST_SRC||' ',
        ' ')-1),'''.')                          (COL(11),3 A);
 END;
        LST_RECPNT      = RECPNT_CORPUS_BI;
        LST_SRC         = CRC_2.SOURCE;
END;
SHUT_2:  CLOSE FILE(WRK1FLE);

/*-----------------------------------------------------------*/
/* CHECKING OF THE NUMBER OF SOURCE-TARGET-PAIRS IN THE      */
/* LEARNING-CORPUS AND - EVENTUALLY - OUTPUT OF AN ERROR-    */
/* MESSAGE:                                                  */
/*-----------------------------------------------------------*/

        IF RECNO_CORPUS>999999
THEN DO;
        CALL LINEFD(57);
        PUT SKIP EDIT(
        '*** THE LEARNING CORPUS CONTAINS MORE THAN 99999',
        '9 SOURCE-',
        '    TARGET-PAIRS.')
        (COL(11),2 A);
END;

/*-----------------------------------------------------------*/
/* FINAL OUTPUT OF A SUMMARIZING MESSAGE CONCERNING THE      */
/* ERROR-MESSAGES:                                           */
/*-----------------------------------------------------------*/

        IF COUNT(SYSPRINT)>0
THEN DO;
        RETURN_CODE = 1;
        CALL LINEFD(57);
```

```
          PUT SKIP EDIT(
          'BECAUSE OF THE ERROR(S) NO DERIVATICN-STATEMENTS',
          ' COULD BE',
          'GENERATED.')
          (COL(11),2 A);
END;
ELSE      PUT SKIP(2) EDIT
          ('THE LEARNING-CORPUS DOES NOT CONTAIN ANY FORMAL',
          ' ERRORS.') (COL(11),2 A);

/*-------------------------------------------------------*/
/* SUBROUTINE-PROCEDURE FCR THE SKIPPING OF A LINE OR FOR */
/* THE DEFINITION OF A NEW PAGE BEFORE A MESSAGE IS WRIT- */
/* TEN ONTO THE SYSPRINT-FILE:                           */
/*-------------------------------------------------------*/

LINEFD:   PROC(MAX_LINENO);
DCL       MAX_LINENO          BIN FIXED(15),
          LINENO              BUILTIN;

          IF LINENO(SYSPRINT)<=MAX_LINENO
           THEN PUT SKIP;
           ELSE SIGNAL ENDPAGE(SYSPRINT);

END       LINEFD;

END       CHKCRPS;
```

```
/*-------------------------------------------------------------*/
/* NAME: DISCOV                                                */
/*                                                             */
/* FUNCTION:                                                   */
/* DISCOV IS A SUBROUTINE-PROCEDURE. IT INFERS DERIVA-         */
/* TION-STATEMENTS FROM A GIVEN LEARNING-CORPUS. THE IN-       */
/* FERENCE IS PERFORMED IN 4 STEPS:                            */
/*  - THE INTERNAL PROCEDURE PREPCRP PREPARES THE LEARN-       */
/*    ING-CORPUS FOR THE INFERENCE OF DERIVATION-STATE-        */
/*    MENTS.                                                   */
/*  - THE INTERNAL PROCEDURE BLCKSTS INFERS BLOCKING-          */
/*    STATEMENTS AND STORES THEM AT THE BEGINNING OF THE       */
/*    STRUCTURE-ARRAY STMT.                                    */
/*  - THE INTERNAL PROCEDURE SBSTSTS INFERS SUBSTITUTION-      */
/*    STATEMENTS AND STORES THEM IN THE STRUCTURE-ARRAY         */
/*    STMT AFTER - POSSIBLY ALREADY INFERED - BLOCKING-        */
/*    STATEMENTS.                                              */
/*  - THE INTERNAL PROCEDURE PREPSTS PREPARES THE INFERED      */
/*    STATEMENTS FOR THEIR FURTHER PROCESSING BY OTHER         */
/*    PROCEDURES.                                              */
/*                                                             */
/* PARAMETER:                                                  */
/*  RETURN_CODE: BIN FIXED(31). IF AN ERROR OCCURS DURING      */
/*               THE EXECUTION OF DISCOV, DISCOV ASSIGNS       */
/*               TO RETURN_CODE AN INTEGER GREATER THAN        */
/*               0. IN DETAIL, THE ASSIGNED INTEGER            */
/*               MEANS:                                        */
/*                 1 - THE GIVEN LEARNING-CORPUS REQUIRES      */
/*                     THE GENERATION OF MORE DERIVATION-      */
/*                     STATEMENTS THAN CAN BE STORED IN        */
/*                     THE STRUCTURE-ARRAY STMT,               */
/*                16 - THE EXECUTION OF THE SORT-UTILITY       */
/*                     SYNCSORT, WHICH IS CALLED BY            */
/*                     DISCOV VIA THE INTERFACE PROGRAM        */
/*                     PLISRTA, FAILED.                        */
/*                                                             */
/* FILES:                                                      */
/*  INPUT:         CORPUS,                                     */
/*  INPUT/OUTPUT: WRK1FLE,                                     */
/*  INPUT/OUTPUT: WRK2FLE,                                     */
/*  INPUT/OUTPUT: WRK3FLE,                                     */
/*  OUTPUT:       SYSPRINT.                                    */
/*    FURTHERMORE THE SORT-UTILITY SYNCSORT, WHICH IS          */
/* CALLED BY DISCOV VIA THE INTERFACE PROGRAM PLISRTA          */
/* NEEDS THE FILES: WRK1IN, WRK1OUT, WRK1WK01, WRK1WK02,       */
/* WRK1WK03, WRK2IN, WRK2OUT, WRK2WK01, WRK2WK02 AND           */
/* WRK2WK03.                                                   */
/*-------------------------------------------------------------*/

DISCOV:  PROC(RETURN_CODE) REORDER;
DCL      1 STMT(*)            CTL EXT,
          2 X_Z               CHAR(30) VAR,
          2 LGT_X             BIN FIXED(15),
          2 Y                 CHAR(30) VAR,
          2 FREQ_SRC          BIN FIXED(31),
```

```
1 BOOKEEP                EXT,
 2 RECNO_CORPUS          BIN FIXED(31),
 2 LST_PRF_BLCK_STMT     BIN FIXED(15) INIT(0),
 2 LST_SUF_BLCK_STMT     BIN FIXED(15) INIT(0),
 2 LST_PRF_SBST_STMT     BIN FIXED(15) INIT(0),
 2 LST_SUF_SBST_STMT     BIN FIXED(15) INIT(0),
 2 DUMMY                 CHAR(58),

1 ARC_1,
 2 PRF,
  3 LGT_POT_X            BIN FIXED(15),
  3 POT_X                CHAR(30),
  3 FREQ_POT_X           BIN FIXED(31),
 2 SUF,
  3 LGT_POT_X            BIN FIXED(15),
  3 POT_X                CHAR(30),
  3 FREQ_POT_X           BIN FIXED(31),
 2 TICK                  BIT(8) INIT('0'B),

1 ARC_2                  BASED(ARC_2_PNTR),
 2 PRF,
  3 LGT_X                BIN FIXED(15),
  3 X                    CHAR(30),
  3 LGT_Y                BIN FIXED(15),
  3 Y                    CHAR(30),
  3 SOURCE               CHAR(30),
 2 SUF,
  3 LGT_X                BIN FIXED(15),
  3 X                    CHAR(29),
  3 LGT_Y                BIN FIXED(15) UNALIGNED,
  3 Y                    CHAR(29),
  3 ISOURCE              CHAR(30),
 2 FREQ_XYP              BIN FIXED(31) UNALIGNED,

WRKCHR                   CHAR(30),
RECNO_WRK1FLE_BI         BIN FIXED(31) INIT(0),
RECNO_WRK1FLE_CH         PIC '999999',
RECNO_WRK2FLE_BI         BIN FIXED(31),
RECNO_WRK2FLE_CH         PIC '999999',
BOUNDARY                 BIN FIXED(15),
MAXSTNO                  BIN FIXED(15) EXT,
STMT_PNTR                BIN FIXED(15) INIT(0),
WRKBIN_1                 BIN FIXED(15),
WRKBIN_2                 BIN FIXED(15),
(I,J,K,L)                BIN FIXED(31),
RETURN_CODE              BIN FIXED(31),
ARC_2_PNTR               PTR,
(HIGH,MAX,MIN)           BUILTIN,
(PLISRTA,SUBSTR)         BUILTIN,
CORPUS                   FILE RECORD SEQI,
WRK1FLE                  FILE RECORD SEQI,
WRK2FLE                  FILE RECORD SEQI,
```

```
         INVERT                ENTRY(CHAR(*),BIN FIXED(15))
                                  RETURNS(CHAR(30) VAR),
         LITMTCH               ENTRY(CHAR(*),CHAR(*),
                                  BIN FIXED(15))
                                  RETURNS(BIN FIXED(15));

         CALL PREPCRP;
            IF RETURN_CODE¬=0 THEN RETURN;
            IF RECNO_WRK1FLE_BI>0 THEN
         CALL BLCKSTS;
            IF RETURN_CODE¬=0 THEN RETURN;
            IF RECNO_WRK2FLE_BI>0 THEN
         CALL SBSTSTS;
            IF RETURN_CODE¬=0 THEN RETURN;
         CALL PREPSTS;

/*-------------------------------------------------------*/
/* PREPARATION OF THE CORPUS FOR THE INFERENCE OF DERIVA- */
/* TION-STATEMENTS:                                      */
/*-------------------------------------------------------*/

PREPCRP: PROC;
DCL      STP                   CHAR(80),
          STP_SOURCE           CHAR(30) DEF STP POS(11),
          STP_TARGET           CHAR(30) DEF STP POS(41),
          CUT_SOURCE(30)       CHAR(1)  DEF STP POS(11),
          CUT_TARGET(30)       CHAR(1)  DEF STP POS(41),

         ARC_1_SOURCE          CHAR(30) DEF PRF.PCT_X,
         ARC_1_ISOURCE         CHAR(30) DEF SUF.PCT_X,
         LST_POT_X             CHAR(30),
         FREQ_POT_X(0:MAXSTNO) BIN FIXED(31),
         LGT_SRC               BIN FIXED(15),
         LGT_TRG               BIN FIXED(15),
         START_IN_SRC          BIN FIXED(15),
         START_IN_TRG          BIN FIXED(15),
         LGT_CMN_STR           BIN FIXED(15),
         (INDEX,LOW)           BUILTIN;

         ALLOCATE ARC_2;
         OPEN FILE(CORPUS)  INPUT,
              FILE(WRK1FLE) OUTPUT,
              FILE(WRK2FLE) OUTPUT;
DO       I=1 TO RECNO_CORPUS;
         READ FILE(CORPUS) INTO(STP);
         LGT_SRC = INDEX(STP_SOURCE||' ',' ')-1;
         LGT_TRG = INDEX(STP_TARGET||' ',' ')-1;
         WRKCHR  = INVERT(STP_SOURCE,LGT_SRC);
```

```
        IF LGT_TRG=0

        /*-----------------------------------------------*/
        /* EACH SOURCE  W I T H O U T  TARGET (=         */
        /* ARC_1_SOURCE) IS STORED WITH ITS INVERSION (= */
        /* ARC_1_ISOURCE) IN WRK1FLE:                    */
        /*-----------------------------------------------*/

THEN DO;
        ARC_1_SOURCE     = STP_SOURCE;
        ARC_1_ISOURCE    = WRKCHR;
        WRITE FILE(WRK1FLE) FROM(ARC_1);
        RECNO_WRK1FLE_BI = RECNO_WRK1FLE_BI+1;
END;

        /*-----------------------------------------------*/
        /* FOR EACH SOURCE  W I T H  TARGET CONSTRUCTION */
        /* OF AN ANALYSIS-RECORD (= ARC_2) FOR THE IN-   */
        /* FERENCE OF SUBSTITUTION-STATEMENTS AND STOR-  */
        /* AGE IN WRK2FLE:                               */
        /*-----------------------------------------------*/

ELSE DO;

             /*-------------------------------------------*/
             /* IDENTIFICATION OF THE FIRST LONGEST COM- */
             /* MON STRING OF EACH SOURCE AND TARGET:    */
             /*-------------------------------------------*/

        START_IN_TRG = LGT_TRG+1;
        START_IN_SRC = LGT_SRC+1;
        LGT_CMN_STR  = 0;
 DO     J=1 TO LGT_TRG WHILE(LGT_TRG-J>=LGT_CMN_STR);
  DO    K=1 TO LGT_SRC WHILE(LGT_SRC-K>=LGT_CMN_STR);
        IF CUT_SOURCE(K)=CUT_TARGET(J)
         & ¬( J-START_IN_TRG=K-START_IN_SRC
            & START_IN_TRG+J<START_IN_TRG+LGT_CMN_STR) THEN
   DO;
        BOUNDARY=MIN(LGT_TRG-J,LGT_SRC-K);
    DO  I=1 TO BOUNDARY;
        IF CUT_SOURCE(K+L)¬=CUT_TARGET(J+L) THEN LEAVE;
    END;
        IF L>LGT_CMN_STR THEN
    DO;
        START_IN_TRG = J;
        START_IN_SRC = K;
        LGT_CMN_STR  = L;
    END;
   END;
  END;
 END;
```

```
            /*-----------------------------------------*/
            /* FOR EACH SOURCE-TARGET-PAIR IDENTIFICA- */
            /* TION OF THE SUBSTITUENDUM AND SUBSTITU- */
            /* ENS OF ITS CORRESPONDING PREFIXAL- AND  */
            /* SUFFIXAL-SUBSTITUTION-STATEMENT:        */
            /*-----------------------------------------*/

          PRF.LGT_X   = START_IN_SRC-1;
          PRF.X       = SUBSTR(STP_SOURCE,1,PRF.LGT_X);
          PRF.LGT_Y   = START_IN_TRG-1;
          PRF.Y       = SUBSTR(STP_TARGET,1,PRF.LGT_Y);
          PRF.SOURCE  = STP_SOURCE;
          SUF.LGT_X   = LGT_SRC-START_IN_SRC-LGT_CMN_STR+1;
          SUF.X       = SUBSTR(WRKCHR,1,SUF.LGT_X);
          SUF.LGT_Y   = LGT_TRG-START_IN_TRG-LGT_CMN_STR+1;
          SUF.Y       = SUBSTR(INVERT(STP_TARGET,-1),1,
                               SUF.LGT_Y);
          SUF.ISOURCE = WRKCHR;
          WRITE FILE(WRK2FLE) FROM(ARC_2);
 END;
END;
          CLOSE FILE(CORPUS),
                FILE(WRK1FLE),
                FILE(WRK2FLE);
          RECNO_WRK2FLE_BI = RECNO_CORPUS-RECNO_WRK1FLE_BI;
          RECNO_WRK1FLE_CH = RECNO_WRK1FLE_BI;
          RECNO_WRK2FLE_CH = RECNO_WRK2FLE_BI;
          IF RECNO_WRK1FLE_BI=0 | RECNO_WRK2FLE_BI=0
           THEN DO;FREE ARC_2;RETURN;END;
```

```
        /*-----------------------------------------------*/
        /* CONSTRUCTION OF AN ANALYSIS-RECORD FOR THE    */
        /* INFERENCE OF BLOCKING-STATEMENTS (= ARC_1)    */
        /* FROM EACH RECORD IN WRK1FLE; STORAGE OF THE   */
        /* RESULT IN WRK1FLE:                            */
        /*-----------------------------------------------*/
             /*-------------------------------------------*/
             /* FOR EACH SOURCE IN WRK1FLE IDENTIFICA-    */
             /* TION OF THE POTENTIAL PREFIXAL IMPEDIENS  */
             /* (= PRF.PCT_X) OF ITS CORRESPONDING        */
             /* BLOCKING-STATEMENT (THIS IMPEDIENS IS     */
             /* THE SHORTEST LEFT SUBSTRING IN WHICH THE  */
             /* SOURCE DIFFERS FROM EACH SOURCE IN        */
             /* WRK2FLE), COMPUTATION OF THE LENGTH OF    */
             /* THIS POTENTIAL IMPEDIENS (= PRF.LGT_POT_  */
             /* X) AND COMPUTATION OF THE FREQUENCY OF    */
             /* THE POTENTIAL IMPEDIENS (= FREQ_POT_X(I))*/
             /* IN ALL SOURCES IN WRK1FLE:                */
             /*-------------------------------------------*/

        CALL PLISRTA(' SORT FIELDS=(3,30,CH,A),FILSZ='||
        RECNO_WRK1FLE_CH||' ',' RECORD TYPE=F,LENGTH=73 ',
        100000,RETURN_CODE,'WRK1');
        IF RETURN_CODE¬=0
         THEN DO;CALL ERRMSG(1);RETURN;END;

        CALL PLISRTA(' SORT FIELDS=(65,30,CH,A),FILSZ='||
        RECNO_WRK2FLE_CH||' ',' RECORD TYPE=F,LENGTH=190 ',
        100000,RETURN_CODE,'WRK2');
        IF RETURN_CODE¬=0
         THEN DO;CALL ERRMSG(2);RETURN;END;

ON      ENDFILE(WRK2FLE) ARC_2.SOURCE = HIGH(1);
        OPEN FILE(WRK1FLE) UPDATE,
             FILE(WRK2FLE) INPUT;
        ARC_2.SOURCE  = LOW(1);
        LST_POT_X     = LOW(1);
        I             = 0;
DO      J=1 TO RECNO_WRK1FLE_BI;
        READ FILE(WRK1FLE) INTO(ARC_1);
 DO     WHILE(ARC_1_SOURCE>ARC_2.SOURCE)
           UNTIL(ARC_1_SOURCE<ARC_2.SOURCE);
        WRKCHR        = ARC_2.SOURCE;
        READ FILE(WRK2FLE) INTO(ARC_2);
 END;
```

```
         PRF.LGT_POT_X =
            MAX(LLTMTCH(WRKCHR,ARC_1_SOURCE,29),
                LLTMTCH(ARC_1_SOURCE,ARC_2.SOURCE,29))+1;
         PRF.POT_X     = SUBSTR(ARC_1_SOURCE,1,
                                PRF.LGT_POT_X);
         REWRITE FILE(WRK1FLE) FROM(ARC_1);
         IF LST_POT_X  = PRF.POT_X
 THEN    FREQ_POT_X(I) = FREQ_POT_X(I)+1;
 ELSE DO;
         I             = I+1;
         FREQ_POT_X(I) = 1;
         LST_POT_X     = PRF.POT_X;
 END;
END;
         CLOSE FILE(WRK1FLE),
               FILE(WRK2FLE);

              /*-----------------------------------------*/
              /* STORAGE OF ITS FREQUENCY WITH EACH PO-  */
              /* TENTIAL PREFIXAL IMPEDIENS IN WRK1FLE:  */
              /*-----------------------------------------*/

         OPEN FILE(WRK1FLE) UPDATE;
DO       J=1 TO I;
 DO      K=1 TO FREQ_POT_X(J);
         READ FILE(WRK1FLE) INTO(ARC_1);
         PRF.FREQ_POT_X = FREQ_POT_X(J);
         REWRITE FILE(WRK1FLE) FROM(ARC_1);
 END;
END;
         CLOSE FILE(WRK1FLE);
```

```
            /*-----------------------------------------*/
            /* FOR EACH SOURCE IN WRK1FLE IDENTIFICA-  */
            /* TION OF THE POTENTIAL SUFFIXAL IMPEDIENS */
            /* (= SUF.POT_X) OF ITS CORRESPONDING      */
            /* BLOCKING-STATEMENT (THIS IMPEDIENS IS   */
            /* THE SHORTEST RIGHT SUBSTRING IN WHICH   */
            /* THE SOURCE DIFFERS FROM EACH SOURCE IN  */
            /* WRK2FLE), COMPUTATION OF THE LENGTH OF  */
            /* THIS POTENTIAL IMPEDIENS (= SUF.LGT_POT_ */
            /* X) AND COMPUTATION OF THE FREQUENCY OF  */
            /* THE POTENTIAL IMPEDIENS (= FREQ_POT_X(I))*/
            /* IN ALL SOURCES IN WRK1FLE:              */
            /*-----------------------------------------*/

         CALL PLISRTA(' SORT FIELDS=(39,30,CH,A),FILSZ='||
         RECNO_WRK1FLE_CH||' ',' RECORD TYPE=F,LENGTH=73 ',
         100000,RETURN_CODE,'WRK1');
         IF RETURN_CODE¬=0
          THEN DO;CALL ERRMSG(1);RETURN;END;

         CALL PLISRTA(' SORT FIELDS=(157,30,CH,A),FILSZ='||
         RECNO_WRK2FLE_CH||' ',' RECORD TYPE=F,LENGTH=190 ',
         100000,RETURN_CODE,'WRK2');
         IF RETURN_CODE¬=0
          THEN DO;CALL ERRMSG(2);RETURN;END;

ON       ENDFILE(WRK2FLE) ARC_2.ISOURCE = HIGH(1);
         OPEN FILE(WRK1FLE) UPDATE,
              FILE(WRK2FLE) INPUT;
         ARC_2.ISOURCE = LOW(1);
         LST_POT_X     = LOW(1);
         I             = 0;
DO       J=1 TO RECNO_WRK1FLE_BI;
         READ FILE(WRK1FLE) INTO(ARC_1);
 DO      WHILE(ARC_1_ISOURCE>ARC_2.ISOURCE)
            UNTIL(ARC_1_ISOURCE<ARC_2.ISOURCE);
         WRKCHR        = ARC_2.ISOURCE;
         READ FILE(WRK2FLE) INTO(ARC_2);
 END;
         SUF.LGT_POT_X =
            MAX(LLTMTCH(WRKCHR,ARC_1_ISOURCE,29),
                LLTMTCH(ARC_1_ISOURCE,ARC_2.ISOURCE,29))+1;
         SUF.POT_X     = SUBSTR(ARC_1_ISOURCE,1,
                                SUF.LGT_POT_X);
         REWRITE FILE(WRK1FLE) FROM(ARC_1);
         IF LST_POT_X  = SUF.POT_X
 THEN    FREQ_POT_X(I) = FREQ_POT_X(I)+1;
 ELSE DO;
         I             = I+1;
         FREQ_POT_X(I) = 1;
         LST_POT_X     = SUF.POT_X;
 END;
```

```
END;
          CLOSE FILE(WRK1FLE),
                FILE(WRK2FLE);

               /*-----------------------------------------*/
               /* STORAGE OF ITS FREQUENCY WITH EACH PO-  */
               /* TENTIAL SUFFIXAL IMPEDIENS IN WRK1FLE:  */
               /*-----------------------------------------*/

          OPEN FILE(WRK1FLE) UPDATE;
DO        J=1 TO I;
 DO       K=1 TO FREQ_POT_X(J);
          READ FILE(WRK1FLE) INTO(ARC_1);
          SUF.FREQ_POT_X = FREQ_POT_X(J);
          REWRITE FILE(WRK1FLE) FROM(ARC_1);
 END;
END;
          CLOSE FILE(WRK1FLE);
          FREE ARC_2;

END       PREPCRP;

/*-----------------------------------------------------------*/
/* INFERENCE OF BLOCKING-STATEMENTS:                         */
/*-----------------------------------------------------------*/

BLCKSTS: PROC;
DCL       LOW                  BUILTIN;

          /*-------------------------------------------------*/
          /* INFERENCE OF BLOCKING-STATEMENTS IF SOURCES     */
          /* WITH TARGETS DO NOT EXIST:                      */
          /*-------------------------------------------------*/

          IF RECNO_WRK2FLE_BI=0
THEN DO;
          STMT_PNTR          = 1;
          X_Z(1)             = '';
          FREQ_SRC(1)        = RECNO_WRK1FLE_BI;
          LST_PRF_BLCK_STMT  = 1;
          RETURN;
END;
```

```
        /*-----------------------------------------------*/
        /* INFERENCE OF BLOCKING-STATEMENTS IF SOURCES    */
        /* WITH TARGETS EXIST:                           */
        /*-----------------------------------------------*/

           STMT.X_Z(0)        = LOW(1);
        CALL PREFBST;
           LST_PRF_BLCK_STMT = STMT_PNTR;
        CALL SUFFBST;
           LST_SUF_BLCK_STMT = STMT_PNTR;
           STMT_PNTR         = MAX(LST_PRF_BLCK_STMT,
                                   LST_SUF_BLCK_STMT);

        /*-----------------------------------------------*/
        /* INFERENCE OF PREFIXAL-BLOCKING-STATEMENTS      */
        /* (BASED ON THE ANALYSIS OF THE DATA IN          */
        /* WRK1FLE):                                     */
        /*-----------------------------------------------*/

PREFBST: PROC;

             /*-----------------------------------------*/
             /* SORTING OF WRK1FLE                      */
             /* 1. ON THE POTENTIAL PREFIXAL IMPEDIENTIA */
             /*    IN ASCENDING ORDER                   */
             /* 2. ON THE FREQUENCY OF THE POTENTIAL    */
             /*    SUFFIXAL IMPEDIENTIA ASCENDING ORDER: */
             /*-----------------------------------------*/

        CALL PLISRTA(' SORT FIELDS=(3,30,CH,A,69,4,FI,A)'||
        ',FILSZ='||RECNO_WRK1FLE_CH||' ',' RECORD TYPE=F'||
        ',LENGTH=73 ',100000,RETURN_CODE,'WRK1');
        IF RETURN_CODE¬=0
         THEN DO;CALL ERRMSG(1);RETURN;END;

        OPEN FILE(WRK1FLE) UPDATE;
DO      I=1 TO RECNO_WRK1FLE_BI;
        READ FILE(WRK1FLE) INTO(ARC_1);
```

```
            /*-----------------------------------------*/
            /* TEST WHETHER THE PREFIXAL-BLOCKING-     */
            /* STATEMENT GENERATED AT LAST WOULD ALSO  */
            /* BE GENERATED FROM THE CURRENT ANALYSIS- */
            /* RECORD (= ARC_1):                       */
            /*-----------------------------------------*/

        IF PRF.POT_X=X_Z(STMT_PNTR)
THEN DO;

            /*-----------------------------------------*/
            /* INCREMENT OF THE COUNTER (= FREQ_SRC(   */
            /* STMT_PNTR)) WHICH INDICATES THE NUMBER  */
            /* OF THE SOURCES IN THE LEARNING-CORPUS TO */
            /* WHICH THE PREFIXAL-BLOCKING-STATEMENT   */
            /* GENERATED AT LAST IS APPLICABLE:        */
            /*-----------------------------------------*/

        FREQ_SRC(STMT_PNTR) = FREQ_SRC(STMT_PNTR)+1;

            /*-----------------------------------------*/
            /* MARKING OF THE CURRENT ANALYSIS-RECORD  */
            /* IN WRK1FLE THAT IT HAS BEEN TREATED BY  */
            /* PREFBST:                                */
            /*-----------------------------------------*/

        TICK = '1'B;
        REWRITE FILE(WRK1FLE) FROM(ARC_1);
END;
ELSE DO;

            /*-----------------------------------------*/
            /* TEST WHETHER A PREFIXAL BLOCKING-STATE- */
            /* MENT SHALL BE GENERATED FROM THE CURRENT */
            /* ANALYSIS-RECORD:                        */
            /*-----------------------------------------*/

        IF PRF.FREQ_POT_X>SUF.FREQ_POT_X
         | ( PRF.FREQ_POT_X=SUF.FREQ_POT_X
           & PRF.LGT_POT_X<=SUF.LGT_POT_X)
 THEN DO;
```

```
            /*-----------------------------------------*/
            /* CONSTRUCTION OF A PREFIXAL-BLOCKING-    */
            /* STATEMENT FROM THE CURRENT ANALYSIS-REC- */
            /* ORD:                                    */
            /*-----------------------------------------*/

         IF STMT_PNTR=MAXSTNO
          THEN DO;CALL ERRMSG(3);RETURN_CODE=1;RETURN;END;
         STMT_PNTR            = STMT_PNTR+1;
         X_Z(STMT_PNTR)       = SUBSTR(PRF.POT_X,1,
                                       PRF.LGT_POT_X);
         FREQ_SRC(STMT_PNTR) = 1;

            /*-----------------------------------------*/
            /* MARKING OF THE CURRENT ANALYSIS-RECORD  */
            /* IN WRK1FLE THAT IT HAS BEEN TREATED BY  */
            /* PREFBST:                                */
            /*-----------------------------------------*/

         TICK                 = '1'B;
         REWRITE FILE(WRK1FLE) FROM(ARC_1);
  END;
 END;
END;
         CLOSE FILE(WRK1FLE);

END      PREFBST;

         /*-------------------------------------------*/
         /* INFERENCE OF SUFFIXAL-BLOCKING-STATEMENTS */
         /* (BASED ON THE ANALYSIS OF THE DATA IN     */
         /* WRK1FLE):                                 */
         /*-------------------------------------------*/

SUFFBST: PROC;
DCL      REL_PNTR             BIN FIXED(15);

            /*-----------------------------------------*/
            /* SORTING OF WRK1FLE ON THE POTENTIAL SUF- */
            /* FIXAL IMPEDIENTIA IN ASCENDING ORDER:   */
            /*-----------------------------------------*/

         CALL PLISRTA(' SORT FIELDS=(39,30,CH,A),FILSZ='||
         RECNO_WRK1FLE_CH||' ',' RECORD TYPE=F,LENGTH=73 ',
         100000,RETURN_CODE,'WRK1');
         IF RETURN_CODE¬=0
          THEN DO;CALL ERRMSG(1);RETURN;END;
```

```
          OPEN FILE(WRK1FLE) INPUT;
          REL_PNTR  = STMT_PNTR;
          STMT_PNTR = 0;
DO        I=1 TO RECNO_WRK1FLE_BI;
          READ FILE(WRK1FLE) INTO(ARC_1);

               /*-----------------------------------------*/
               /* TEST WHETHER THE CURRENT ANALYSIS-RECORD */
               /* (= ARC_1) HAS ALREADY BEEN TREATED BY    */
               /* PREFBST:                                 */
               /*-----------------------------------------*/

          IF TICK='0'B
 THEN DO;

               /*-----------------------------------------*/
               /* TEST WHETHER THE SUFFIXAL-BLOCKING-      */
               /* STATEMENT GENERATED AT LAST WOULD ALSO   */
               /* BE GENERATED FROM THE CURRENT ANALYSIS-  */
               /* RECORD:                                  */
               /*-----------------------------------------*/

          IF SUF.POT_X=X_Z(STMT_PNTR)

               /*-----------------------------------------*/
               /* INCREMENT OF THE COUNTER (= FREQ_SRC(    */
               /* STMT_PNTR)) WHICH INDICATES THE NUMBER   */
               /* OF THE SOURCES IN THE LEARNING-CORPUS TO */
               /* WHICH THE SUFFIXAL-BLOCKING-STATEMENT    */
               /* GENERATED AT LAST IS APPLICABLE:         */
               /*-----------------------------------------*/

  THEN    FREQ_SRC(STMT_PNTR) = FREQ_SRC(STMT_PNTR)+1;

               /*-----------------------------------------*/
               /* CONSTRUCTION OF A SUFFIXAL-BLOCKING-     */
               /* STATEMENT FROM THE CURRENT ANALYSIS-REC- */
               /* ORD:                                     */
               /*-----------------------------------------*/

  ELSE DO;
          IF REL_PNTR=MAXSTNO
           THEN DO;CALL ERRMSG(3);RETURN_CODE=1;RETURN;END;
          REL_PNTR,STMT_PNTR  = REL_PNTR+1;
          X_Z(STMT_PNTR)      = SUBSTR(SUF.POT_X,1,
                                       SUF.LGT_POT_X);
          FREQ_SRC(STMT_PNTR) = 1;
  END;
 END;
END;
          CLOSE FILE(WRK1FLE);
```

```
END      SUFFBST;

END      BLCKSTS;

/*-------------------------------------------------------*/
/* INFERENCE OF SUBSTITUTION-STATEMENTS:                 */
/*-------------------------------------------------------*/

SBSTSTS: PROC;
DCL      LST_X                  CHAR(30),
         LST_Y                  CHAR(30),
         REL_PNTR               BIN FIXED(15),
         REC_PNTR1              BIN FIXED(20),
         REC_PNTR2              BIN FIXED(20),
         L_BDRY                 BIN FIXED(20),
         U_BDRY                 BIN FIXED(20),
         MIDDLE                 BIN FIXED(20),
         FREQ_XYP(0:MAXSTNO)    BIN FIXED(31),
         FRST_ARC(0:MAXSTNO)    BIN FIXED(31) DEF FREQ_XYP,
         RECNO_WRK3FLE_BI       BIN FIXED(31),
         LENGTH                 BUILTIN,
         WRK3FLE                FILE RECORD KEYED
                                   ENV(REGIONAL(1));

            RECNO_WRK3FLE_BI = RECNO_WRK2FLE_BI-1;
            STMT.LGT_X(0)    = -1;
         CALL PREFSST;
            IF RETURN_CODE¬=0 THEN RETURN;
         CALL SUFFSST;

         /*-------------------------------------------*/
         /* INFERENCE OF PREFIXAL-SUBSTITUTION-STATE- */
         /* MENTS:                                    */
         /*-------------------------------------------*/

PREFSST: PROC;

              /*-----------------------------------*/
              /* SORTING OF WRK2FLE ON THE PREFIXAL SUB- */
              /* STITUENDA AND SUBSTITUENTIA IN ASCENDING */
              /* ORDER:                            */
              /*-----------------------------------*/

         CALL PLISRTA(' SORT FIELDS=(3,30,CH,A,35,30,CH,A'||
         '),FILSZ='||RECNO_WRK2FLE_CH||' ',' RECORD TYPE='||
         'F,LENGTH=190 ',100000,RETURN_CODE,'WRK2');
```

```
          IF RETURN_CODE¬=0
           THEN DO;CALL ERRMSG(2);RETURN;END;

               /*-----------------------------------------*/
               /* IDENTIFICATION OF THE ABSOLUTE FREQUENCY */
               /* OF EACH PREFIXAL SUBSTITUENDUM-SUBSTITU- */
               /* ENS-PAIR IN WRK2FLE AND STORAGE OF THIS  */
               /* FREQUENCY WITH EACH SUCH PAIR IN         */
               /* WRK2FLE:                                 */
               /*-----------------------------------------*/

          OPEN FILE(WRK2FLE) INPUT;
          LST_X,LST_Y = HIGH(1);
          J           = 0;
DO        I=1 TO RECNO_WRK2FLE_BI;
          READ FILE(WRK2FLE) SET(ARC_2_PNTR);
          IF PRF.X=LST_X & PRF.Y=LST_Y
 THEN     FREQ_XYP(J) = FREQ_XYP(J)+1;
 ELSE DO;
          J           = J+1;
          FREQ_XYP(J) = 1;
          LST_X       = PRF.X;
          LST_Y       = PRF.Y;
 END;
END;
          CLOSE FILE(WRK2FLE);
          OPEN FILE(WRK2FLE) UPDATE;
DO        I=1 TO J;
 DO       K=1 TO FREQ_XYP(I);
          READ FILE(WRK2FLE) SET(ARC_2_PNTR);
          ARC_2.FREQ_XYP = FREQ_XYP(I);
          REWRITE FILE(WRK2FLE) FROM(ARC_2);
 END;
END;
          CLOSE FILE(WRK2FLE);
```

```
        /*-----------------------------------------*/
        /* SORTING OF WRK2FLE                      */
        /* 1. ON THE ABSOLUTE FREQUENCY OF EACH    */
        /*    PREFIXAL SUBSTITUENDUM-SUBSTITUENS-  */
        /*    PAIR IN DESCENDING ORDER.            */
        /* 2. ON THE PREFIXAL SUBSTITUENDA AND SUB- */
        /*    STITUENTIA AND ON THE SOURCES IN DE-  */
        /*    CENDING ORDER:                       */
        /*-----------------------------------------*/

      CALL PLISRTA(' SORT FIELDS=(187,4,FI,D,3,30,CH,D'||
      ',35,60,CH,D),FILSZ='||RECNO_WRK2FLE_CH||' ',
      ' RECORD TYPE=F,LENGTH=190 ',100000,RETURN_CODE,
      'WRK2');
      IF RETURN_CODE¬=0
       THEN DO;CALL ERRMSG(2);RETURN;END;

        /*-----------------------------------------*/
        /* COPYING OF WRK2FLE INTO THE DIRECT-AC-  */
        /* CESS-DATA-SET WRK3FLE:                  */
        /*-----------------------------------------*/

      OPEN FILE(WRK2FLE) INPUT,
         FILE(WRK3FLE) SEQL OUTPUT;
DO    REC_PNTR1=0 TO RECNO_WRK3FLE_BI;
      READ FILE(WRK2FLE) SET(ARC_2_PNTR);
      WRITE FILE(WRK3FLE) FROM(ARC_2) KEYFROM(REC_PNTR1);
END;
      CLOSE FILE(WRK2FLE),
            FILE(WRK3FLE);

        /*-----------------------------------------*/
        /* GENERATION OF THE PREFIXAL-SUBSTITUTION- */
        /* STATEMENTS (BASED ON THE ANALYSIS OF THE */
        /* DATA IN WRK3FLE):                       */
        /*-----------------------------------------*/
            /*-------------------------------------*/
            /* INITIALIZATIONS:                    */
            /*-------------------------------------*/

      ALLOCATE ARC_2;
      OPEN FILE(WRK3FLE) DIRECT INPUT;
      REL_PNTR      = STMT_PNTR;
      J             = STMT_PNTR+1;
      STMT_PNTR     = 0;

DO    REC_PNTR1=0 TO RECNO_WRK3FLE_BI;
      READ FILE(WRK3FLE) INTO(ARC_2) KEY(REC_PNTR1);
```

```
                /*-------------------------------------*/
                /* TEST WHETHER THE PREFIXAL-SUBSTITU- */
                /* TION-STATEMENT GENERATED AT LAST    */
                /* WOULD ALSO BE GENERATED FROM THE    */
                /* CURRENT ANALYSIS-RECORD:            */
                /*-------------------------------------*/

        IF ¬( STMT.LGT_X(STMT_PNTR) = PRF.LGT_X
            & X_Z(STMT_PNTR)        = SUBSTR(PRF.SOURCE,1,
                                      LENGTH(
                                      X_Z(STMT_PNTR)))
            & STMT.Y(STMT_PNTR)     = PRF.Y)
THEN DO;

                /*-------------------------------------*/
                /* DETERMINATION OF THE NUMBER OF      */
                /* SOURCES IN THE LEARNING-CORPUS TO   */
                /* WHICH THE PREFIXAL-SUBSTITUTION-    */
                /* STATEMENT GENERATED AT LAST IS AP-  */
                /* PLICABLE:                           */
                /*-------------------------------------*/

        FREQ_SRC(STMT_PNTR)   = REC_PNTR1-
                                FRST_ARC(STMT_PNTR);

                /*-------------------------------------*/
                /* DETERMINATION OF SUBSTITUENS AND    */
                /* SUBSTITUENDUM OF THE NEXT PREFIXAL- */
                /* SUBSTITUTION-STATEMENT:             */
                /*-------------------------------------*/

        IF REL_PNTR=MAXSTNO
         THEN DO;CALL ERRMSG(3);RETURN_CODE=1;RETURN;END;
        REL_PNTR,STMT_PNTR    = REL_PNTR+1;
        X_Z(STMT_PNTR)        = SUBSTR(PRF.X,1,PRF.LGT_X);
        STMT.LGT_X(STMT_PNTR) = PRF.LGT_X;
        STMT.Y(STMT_PNTR)     = SUBSTR(PRF.Y,1,PRF.LGT_Y);
        FRST_ARC(STMT_PNTR)   = REC_PNTR1;

                /*-------------------------------------*/
                /* DETERMINATION OF THE CONTEXTUAL-    */
                /* STRING:                             */
                /*-------------------------------------*/

        WRKCHR                = PRF.SOURCE;
        BOUNDARY              = PRF.LGT_X;
GET_XZ: DO I=STMT_PNTR-1 TO J BY -1;
```

```
        IF LLTMTCH(X_Z(STMT_PNTR),X_Z(I),0)=MIN(BOUNDARY,
                                              LENGTH(X_Z(I)))
         THEN IF STMT.LGT_X(STMT_PNTR)¬=STMT.LGT_X(I)
              | STMT.Y(STMT_PNTR)     ¬=STMT.Y(I)
   THEN DO;

                        /*-------------------------------*/
                        /* THE FOLLOWING BINARY-SEARCH-  */
                        /* ALGORITHM HAS - SLIGHTLY MODI-*/
                        /* FIED - BEEN TAKEN FROM: KNUTH,*/
                        /* D. E. (1973): THE ART OF COM- */
                        /* PUTER PROGRAMMING. VOL. 3.    */
                        /* SORTING AND SEARCHING. READING*/
                        /* (MASS.)/MENLO PARK (CAL.)/    */
                        /* LONDON/AMSTERDAM/DON MILLS    */
                        /* (ONTARIO)/SYDNEY. P. 407-411. */
                        /*-------------------------------*/

        L_BDRY   = FRST_ARC(I);
        U_BDRY   = FRST_ARC(I)+FREQ_SRC(I)-1;
    DO  WHILE(U_BDRY-L_BDRY>1);
        MIDDLE   = (L_BDRY+U_BDRY)/2;
        READ FILE(WRK3FLE) INTO(ARC_2) KEY(MIDDLE);
        IF WRKCHR<PRF.SOURCE
         THEN L_BDRY = MIDDLE;
         ELSE U_BDRY = MIDDLE;
    END;
        WRKBIN_1 = 0;
    DO  REC_PNTR2=L_BDRY TO U_BDRY;
        READ FILE(WRK3FLE) INTO(ARC_2) KEY(REC_PNTR2);
        WRKBIN_2 = LLTMTCH(WRKCHR,PRF.SOURCE,29);
        IF WRKBIN_2>WRKBIN_1 THEN WRKBIN_1 = WRKBIN_2;
    END;
        IF WRKBIN_1>=BOUNDARY
    THEN DO;
        BOUNDARY       = WRKBIN_1+1;
        X_Z(STMT_PNTR) = SUBSTR(WRKCHR,1,BOUNDARY);
    END;
        IF BOUNDARY>LENGTH(X_Z(I)) THEN LEAVE GET_XZ;
   END;
  END;
 END;
END;
        FREQ_SRC(STMT_PNTR) = REC_PNTR1-
                              FRST_ARC(STMT_PNTR);
        LST_PRF_BEST_STMT   = STMT_PNTR;
        CLOSE FILE(WRK3FLE);
        FREE ARC_2;
```

```
END      PREFSST;

         /*-----------------------------------------------*/
         /* INFERENCE OF SUFFIXAL-SUBSTITUTION-STATE-     */
         /* MENTS:                                        */
         /*-----------------------------------------------*/

SUFFSST: PROC;

              /*-----------------------------------------*/
              /* SORTING OF WRK2FLE ON THE SUFFIXAL SUB- */
              /* STITUENDA AND SUBSTITUENTIA IN ASCENDING */
              /* ORDER:                                  */
              /*-----------------------------------------*/

         CALL PLISRTA(' SORT FIELDS=(97,29,CH,A,128,29,'||
         'CH,A),FILSZ='||RECNO_WRK2FLE_CH||' ',
         ' RECORD TYPE=F,LENGTH=190 ',100000,RETURN_CODE,
         'WRK2');
         IF RETURN_CODE¬=0
          THEN DO;CALL ERRMSG(2);RETURN;END;

              /*-----------------------------------------*/
              /* IDENTIFICATION OF THE ABSOLUTE FREQUENCY */
              /* OF EACH SUFFIXAL SUBSTITUENDUM-SUBSTITU- */
              /* ENS PAIR IN WRK2FLE AND STORAGE OF THIS  */
              /* FREQUENCY WITH EACH SUCH PAIR IN        */
              /* WRK2FLE:                                */
              /*-----------------------------------------*/

         OPEN FILE(WRK2FLE) INPUT;
         LST_X,LST_Y = HIGH(1);
         J           = 0;
DO       I=1 TO RECNO_WRK2FLE_BI;
         READ FILE(WRK2FLE) SET(ARC_2_PNTR);
         IF SUF.X=LST_X & SUF.Y=LST_Y
 THEN    FREQ_XYP(J) = FREQ_XYP(J)+1;
 ELSE DO;
         J           = J+1;
         FREQ_XYP(J) = 1;
         LST_X       = SUF.X;
         LST_Y       = SUF.Y;
 END;
END;
         CLOSE FILE(WRK2FLE);
         OPEN FILE(WRK2FLE) UPDATE;
DO       I=1 TO J;
 DO      K=1 TO FREQ_XYP(I);
         READ FILE(WRK2FLE) SET(ARC_2_PNTR);
         ARC_2.FREQ_XYP = FREQ_XYP(I);
         REWRITE FILE(WRK2FLE) FROM(ARC_2);
 END;
```

```
END;
          CLOSE FILE(WRK2FLE);

               /*-----------------------------------------*/
               /* SORTING OF WRK2FLE                      */
               /* 1. ON THE ABSOLUTE FREQUENCY OF EACH    */
               /*    SUFFIXAL SUBSTITUENDUM-SUBSTITUENS-  */
               /*    PAIR IN DESCENDING ORDER             */
               /* 2. ON THE SUFFIXAL SUBSTITUENDA AND SUB- */
               /*    STITUENTIA AND ON THE INVERTED SOURC- */
               /*    ES IN ASCENDING ORDER:               */
               /*-----------------------------------------*/

          CALL PLISRTA(' SORT FIELDS=(187,4,FI,D,97,29,CH,'||
          'A,128,59,CH,A),FILSZ='||RECNO_WRK2FLE_CH||' ',
          ' RECORD TYPE=F,LENGTH=190 ',100000,RETURN_CODE,
          'WRK2');
          IF RETURN_CODE¬=0
           THEN DO;CALL ERRMSG(2);RETURN;END;

               /*-----------------------------------------*/
               /* COPYING OF WRK2FLE INTO THE DIRECT-AC-  */
               /* CESS-DATA-SET WRK3FLE:                  */
               /*-----------------------------------------*/

          OPEN FILE(WRK2FLE) INPUT,
               FILE(WRK3FLE) SEQL OUTPUT;
DO        REC_PNTR1=0 TO RECNO_WRK3FLE_BI;
          READ FILE(WRK2FLE) SET(ARC_2_PNTR);
          WRITE FILE(WRK3FLE) FROM(ARC_2) KEYFROM(REC_PNTR1);
END;
          CLOSE FILE(WRK2FLE),
                FILE(WRK3FLE);

               /*-----------------------------------------*/
               /* GENERATION OF THE SUFFIXAL-SUBSTITUTION- */
               /* STATEMENTS (BASED ON THE ANALYSIS OF THE */
               /* DATA IN WRK3FLE):                       */
               /*-----------------------------------------*/
                    /*----------------------------------*/
                    /* INITIALIZATIONS:                 */
                    /*----------------------------------*/

          ALLOCATE ARC_2;
          OPEN FILE(WRK3FLE) DIRECT INPUT;
          REL_PNTR       = STMT_PNTR;
          J              = STMT_PNTR+1;
          STMT_PNTR      = 0;
```

```
DO      REC_PNTR1=0 TO RECNO_WRK3FLE_BI;
        READ FILE(WRK3FLE) INTO(ARC_2) KEY(REC_PNTR1);

                /*-------------------------------------*/
                /* TEST WHETHER THE SUFFIXAL-SUBSTITU- */
                /* TION-STATEMENT GENERATED AT LAST    */
                /* WOULD ALSO BE GENERATED FROM THE    */
                /* CURRENT ANALYSIS-RECORD:            */
                /*-------------------------------------*/

        IF ¬( STMT.LGT_X(STMT_PNTR) = SUF.LGT_X
            & X_Z(STMT_PNTR)        = SUBSTR(SUF.ISOURCE,1,
                                      LENGTH(
                                      X_Z(STMT_PNTR)))
            & STMT.Y(STMT_PNTR)     = SUF.Y)
 THEN DO;

                /*-------------------------------------*/
                /* DETERMINATION OF THE NUMBER OF      */
                /* SOURCES IN THE LEARNING-CORPUS TO   */
                /* WHICH THE SUFFIXAL-SUBSTITUTION-    */
                /* STATEMENT GENERATED AT LAST IS AP-  */
                /* PLICABLE:                           */
                /*-------------------------------------*/

        FREQ_SRC(STMT_PNTR)   = REC_PNTR1-
                                FRST_ARC(STMT_PNTR);

                /*-------------------------------------*/
                /* DETERMINATION OF SUBSTITUENS AND    */
                /* SUBSTITUENDUM OF THE NEXT SUFFIXAL- */
                /* SUBSTITUTION-STATEMENT:             */
                /*-------------------------------------*/

        IF REL_PNTR=MAXSINO
         THEN DO;CALL ERRMSG(3);RETURN_CODE=1;RETURN;END;
        REL_PNTR,STMT_PNTR    = REL_PNTR+1;
        X_Z(STMT_PNTR)        = SUBSTR(SUF.X,1,SUF.LGT_X);
        STMT.LGT_X(STMT_PNTR) = SUF.LGT_X;
        STMT.Y(STMT_PNTR)     = SUBSTR(SUF.Y,1,SUF.LGT_Y);
        FRST_ARC(STMT_PNTR)   = REC_PNTR1;
```

```
                    /*-----------------------------------*/
                    /* DETERMINATION OF THE CONTEXTUAL-  */
                    /* STRING:                           */
                    /*-----------------------------------*/

         WRKCHR                 = SUF.ISOURCE;
         BOUNDARY               = SUF.LGT_X;
  GET_XZ: DO I=STMT_PNTR-1 TO J BY -1;
         IF LLTMTCH(X_Z(STMT_PNTR),X_Z(I),0)=MIN(BOUNDARY,
                                              LENGTH(X_Z(I)))
          THEN IF STMT.LGT_X(STMT_PNTR)¬=STMT.LGT_X(I)
               | STMT.Y(STMT_PNTR)    ¬=STMT.Y(I)
   THEN DO;

                        /*-------------------------------*/
                        /* THE FOLLOWING BINARY-SEARCH-  */
                        /* ALGORITHM HAS - SLIGHTLY MODI-*/
                        /* FIED - BEEN TAKEN FROM: KNUTH,*/
                        /* D. E. (1973): THE ART OF COM- */
                        /* PUTER PROGRAMMING. VOL. 3.    */
                        /* SORTING AND SEARCHING. READING*/
                        /* (MASS.)/MENLO PARK (CAL.)/    */
                        /* LONDON/AMSTERDAM/DON MILLS    */
                        /* (ONTARIO)/SYDNEY. P. 407-411. */
                        /*-------------------------------*/

         L_BDRY   = FRST_ARC(I);
         U_BDRY   = FRST_ARC(I)+FREQ_SRC(I)-1;
    DO   WHILE(U_BDRY-L_BDRY>1);
         MIDDLE   = (L_BDRY+U_BDRY)/2;
         READ FILE(WRK3FLE) INTO(ARC_2) KEY(MIDDLE);
         IF WRKCHR<SUF.ISOURCE
          THEN U_BDRY = MIDDLE;
          ELSE L_BDRY = MIDDLE;
    END;
         WRKBIN_1 = 0;
    DO   REC_PNTR2=L_BDRY TO U_BDRY;
         READ FILE(WRK3FLE) INTO(ARC_2) KEY(REC_PNTR2);
         WRKBIN_2 = LLTMTCH(WRKCHR,SUF.ISOURCE,29);
         IF WRKBIN_2>WRKBIN_1 THEN WRKBIN_1 = WRKBIN_2;
    END;
         IF WRKBIN_1>=BOUNDARY
    THEN DO;
         BOUNDARY       = WRKBIN_1+1;
         X_Z(STMT_PNTR) = SUBSTR(WRKCHR,1,BOUNDARY);
    END;
         IF BOUNDARY>LENGTH(X_Z(I)) THEN LEAVE GET_XZ;
   END;
  END;
 END;
END;
```

```
          FREQ_SRC(STMT_PNTR) = REC_PNTR1-
                                FRST_ARC(STMT_PNTR);
          LST_SUF_SBST_STMT   = STMT_PNTR;
          CLOSE FILE(WRK3FLE);
          FREE ARC_2;

END       SUFFSST;

END       SBSTSTS;

/*-----------------------------------------------------------*/
/* PREPARATION OF THE GENERATED DERIVATION-STATEMENTS FOR */
/* THE FURTHER PROCESSING BY OTHER PROCEDURES. THE PREPA- */
/* RATION CONSISTS IN INVERTING THE ORDER OF THE PREFIX-  */
/* AL-SUBSTITUTION-STATEMENTS AND THE ORDER OF THE SUF-   */
/* FIXAL-SUBSTITUTION-STATEMENTS:                         */
/*-----------------------------------------------------------*/

PREPSTS: PROC;

          /*-----------------------------------------------*/
          /* PREPARATION OF THE PREFIXAL-SUBSTITUTION-     */
          /* STATEMENTS:                                   */
          /*-----------------------------------------------*/

          CALL SWAP(MAX(LST_PRF_BLCK_STMT,LST_SUF_BLCK_STMT)
                    +1,LST_PRF_SBST_STMT);

          /*-----------------------------------------------*/
          /* PREPARATION OF THE SUFFIXAL-SUBSTITUTION-     */
          /* STATEMENTS:                                   */
          /*-----------------------------------------------*/

          CALL SWAP(LST_PRF_SBST_STMT+1,LST_SUF_SBST_STMT);

          /*-----------------------------------------------*/
          /* INVERSION OF THE ORDER OF ALL DERIVATION-     */
          /* STATEMENTS IN STMT WITH A SUBSCRIPT GREATER   */
          /* THAN OR EQUAL TO START AND LESS THAN OR EQUAL */
          /* TO LIMIT:                                     */
          /*-----------------------------------------------*/

SWAP:     PROC(START,LIMIT);
DCL       START                BIN FIXED(15),
          LIMIT                BIN FIXED(15);

          WRKBIN_1        = START;
          WRKBIN_2        = LIMIT;
DO        WHILE(WRKBIN_2>WRKBIN_1);
          STMT(0)         = STMT(WRKBIN_2);
          STMT(WRKBIN_2)  = STMT(WRKBIN_1);
          STMT(WRKBIN_1)  = STMT(0);
          WRKBIN_1        = WRKBIN_1+1;
```

```
          WRKBIN_2        = WRKBIN_2-1;
END;

END       SWAP;

END       PREPSTS;

/*-------------------------------------------------------*/
/* OUTPUT OF ERROR-MESSAGES ONTO THE SYSPRINT-FILE:      */
/*-------------------------------------------------------*/

ERRMSG:   PROC(MSG_NO);
DCL       MSG_NO                BIN FIXED(15),
          MSG(3)                LABEL,
          (CHAR,VERIFY)         BUILTIN,
          SYSPRINT              FILE STREAM PRINT;

          SIGNAL ENDPAGE(SYSPRINT);
          GOTO MSG(MSG_NO);

MSG(1):   PUT SKIP EDIT(
          '*** ERROR DURING SORTING OF WRK1FLE AND BREAK OF',
          ' PRISM.',
          '    PLEASE CHECK THE ALLOCATION OF DATA SETS TO ',
          'THE FILES',
          '    WRK1IN, WRK1OUT, WRK1WK01, WRK1WK02 AND ',
          'WRK1WK03. MAKE',
          '    SURE THAT THESE DATA SETS HAVE APPROPRIATE ',
          'ATTRIBUTES',
          '    AND ARE ALLOCATED WITH SUFFICIENT SPACE.')
          (COL(11),2 A);
          RETURN;

MSG(2):   PUT SKIP EDIT(
          '*** ERROR DURING SORTING OF WRK2FLE AND BREAK OF',
          ' PRISM.',
          '    PLEASE CHECK THE ALLOCATION OF DATA SETS TO ',
          'THE FILES',
          '    WRK2IN, WRK2OUT, WRK2WK01, WRK2WK02 AND ',
          'WRK2WK03. MAKE',
          '    SURE THAT THESE DATA SETS HAVE APPROPRIATE ',
          'ATTRIBUTES',
          '    AND ARE ALLOCATED WITH SUFFICIENT SPACE.')
          (COL(11),2 A);
          RETURN;
```

```
MSG(3):  PUT SKIP EDIT
         ('*** ERROR DURING THE GENERATION OF DERIVATION-',
         'STATEMENTS')                (COL(11),2 A)
         ('    AND BREAK OF PRISM. THE GIVEN LEARNING-COR',
         'PUS REQUIRES')              (COL(11),2 A)
         ('    THE GENERATION OF MORE THAN ',
         SUBSTR(CHAR(MAXSTNO),VERIFY(CHAR(MAXSTNO),' ')),
         ' (INTERNAL) DERIVATION-')   (COL(11),3 A)
         ('    STATEMENTS. PRISM, HOWEVER, CAN GENERATE ',
         'ONLY UP TO')                (COL(11),2 A)
         (SUBSTR(CHAR(MAXSTNO),VERIFY(CHAR(MAXSTNO),' ')),
         ' STATEMENTS FOR A LEARNING-CORPUS.')
                                      (COL(15),2 A);
         RETURN;

END      ERRMSG;

END      DISCOV;
```

```
/*-------------------------------------------------------------*/
/* NAME: STMTOUT                                               */
/*                                                             */
/* FUNCTION:                                                   */
/* STMTOUT IS A SUBROUTINE-PROCEDURE. IT TRANSFORMS A SET      */
/* OF DERIVATION-STATEMENTS INTO A NOTATION USUALLY USED       */
/* BY LINGUISTS. THESE TRANSFORMED STATEMENTS ARE THEN         */
/* WRITTEN WITH THEIR NUMBER AND THE NUMBER OF THE             */
/* SOURCE-TARGET-PAIRS FROM WHICH THEY WERE INFERED ONTO       */
/* THE SYSPRINT-FILE. OPTIONALLY, STMTOUT COMPUTES FOR         */
/* EACH STATEMENT THE ABSOLUTE AND RELATIVE (IN PER CENT)      */
/* FREQUENCIES OF THE SOURCES IN THE CORRESPONDING LEARN-      */
/* ING-CORPUS TO WHICH THE STATEMENT IS APPLICABLE, AND        */
/* IT WRITES THESE FREQUENCIES WITH EACH STATEMENT ONTO        */
/* THE SYSPRINT-FILE.                                          */
/*                                                             */
/* PARAMETER:                                                  */
/*  STATISTICS: BIT(1).                                        */
/*              - IF STATISTICS='1'B, STMTOUT COMPUTES         */
/*                FOR EACH STATEMENT THE FREQUENCIES (S.       */
/*                A.) AND WRITES THEM ONTO THE SYSPRINT-       */
/*                FILE.                                        */
/*              - IF STATISTICS='0'B, STMTOUT DOES NOT         */
/*                COMPUTE THE FREQUENCIES.                     */
/*                                                             */
/* FILE:                                                       */
/*  OUTPUT: SYSPRINT.                                          */
/*-------------------------------------------------------------*/

STMTOUT: PROC(STATISTICS) REORDER;
DCL      1 STMT(*)              CTL EXT,
          2 X_Z                 CHAR(30) VAR,
          2 LGT_X               BIN FIXED(15),
          2 Y                   CHAR(30) VAR,
          2 FREQ_SRC            BIN FIXED(31),

         1 BOOKEEP              EXT,
          2 RECNO_CORPUS        BIN FIXED(31),
          2 LST_PRF_BLCK_STMT   BIN FIXED(15),
          2 LST_SUF_BLCK_STMT   BIN FIXED(15),
          2 LST_PRF_SBST_STMT   BIN FIXED(15),
          2 LST_SUF_SBST_STMT   BIN FIXED(15),
          2 DUMMY               CHAR(58),

         INDENT                 CHAR(22) VAR,
         WRKCHR                 CHAR(60) VAR,
         X                      CHAR(30) VAR,
         Z                      CHAR(32) VAR,
         STATISTICS             BIT(1),
         TITLE                  BIT(1) INIT('1'B),
         PAGENO                 BIN FIXED(15) EXT,
         STMT_COUNTER           BIN FIXED(15),
         (I,J,K)                BIN FIXED(15),
         FREQ_SRC_EQ_XY         BIN FIXED(31),
         (LENGTH,LINENO,MAX)    BUILTIN,
         (SIGN,SUBSTR)          BUILTIN,
```

```
         SYSPRINT              FILE STREAM PRINT,
         INVERT                ENTRY(CHAR(*),BIN FIXED(15))
                                  RETURNS(CHAR(30) VAR);

/*-------------------------------------------------------*/
/* ENDPAGE-CONDITION-ON-UNIT FOR THE OUTPUT OF A HEADING */
/* AT THE START OF A NEW PAGE:                           */
/*-------------------------------------------------------*/

ON       ENDPAGE(SYSPRINT) BEGIN;
         PAGENO = PAGENO+1;
         PUT PAGE EDIT('PAGE ',PAGENO) (COL(62),A,F(4));
         IF TITLE
          THEN PUT SKIP(3) EDIT
               ('GENERATED DERIVATION-STATEMENTS:',(32)'-')
               (COL(11),A);
         PUT SKIP(2);
         IF STATISTICS
          THEN PUT SKIP EDIT(
               'SOURCES IN THE   |',
               'LEARNING-CORPUS  |',
               'TO WHICH THE     |           STATEMENT',
               'STATEMENT IS AP- |',
               'PLICABLE         |',
               (8)'-'||'+'||(8)'-'||'+',
               ' P. C.  |  ABS.  |',
               (8)'-'||'+'||(8)'-'||'+'||(42)'-',
               (8)' '||'|'||(8)' '||'|')
               (COL(11),A);
END;

         SIGNAL ENDPAGE(SYSPRINT);
         TITLE = '0'B;

/*-------------------------------------------------------*/
/* OUTPUT OF BLOCKING-STATEMENTS:                        */
/*-------------------------------------------------------*/
         /*-----------------------------------------------*/
         /* OUTPUT OF PREFIXAL-BLOCKING-STATEMENTS:       */
         /*-----------------------------------------------*/

DO       I=1 TO LST_PRF_BLCK_STMT;
         IF SUBSTR(X_Z(I),LENGTH(X_Z(I)))=' '
          THEN WRKCHR=''''||SUBSTR(X_Z(I),1,LENGTH(X_Z(I))
                    -1)||'''#';
          ELSE WRKCHR=''''||X_Z(I)||'''';
```

```
          IF STATISTICS
           THEN PUT SKIP EDIT
                (100.000*FREQ_SRC(I)/RECNO_CORPUS,'% | ',
                FREQ_SRC(I),' | #',WRKCHR,'¬')
                (COL(11),F(6,2),A,F(6),3 A);
           ELSE PUT SKIP EDIT
                ('#',WRKCHR,'¬') (COL(11),3 A);
END;
          STMT_COUNTER = LST_PRF_BLCK_STMT;

          /*-----------------------------------------------*/
          /* OUTPUT OF SUFFIXAL-BLOCKING-STATEMENTS:       */
          /*-----------------------------------------------*/

          CALL LINEFD(LST_SUF_BLCK_STMT);
DO        I=LST_PRF_BLCK_STMT+1 TO LST_SUF_BLCK_STMT;
          IF SUBSTR(X_Z(I),LENGTH(X_Z(I)))=' '
           THEN WRKCHR='#'''||INVERT(SUBSTR(X_Z(I),1,LENGTH(
                                 X_Z(I))-1),0)||'''';
           ELSE WRKCHR=''''||INVERT(X_Z(I),0)||'''';
          IF STATISTICS
           THEN PUT SKIP EDIT
                (100.000*FREQ_SRC(I)/RECNO_CORPUS,'% | ',
                FREQ_SRC(I),' | ',WRKCHR,'#¬')
                (COL(11),F(6,2),A,F(6),3 A);
           ELSE PUT SKIP EDIT
                (WRKCHR,'#¬') (COL(11),2 A);
END;
          STMT_COUNTER = MAX(LST_PRF_BLCK_STMT,
                             LST_SUF_BLCK_STMT);

/*-----------------------------------------------------------*/
/* OUTPUT OF SUBSTITUTION-STATEMENTS:                        */
/*-----------------------------------------------------------*/

          IF STATISTICS
           THEN INDENT = (8)' '||'|'||(8)' '||'|    ';
           ELSE INDENT = '   ';

          /*-----------------------------------------------*/
          /* OUTPUT OF PREFIXAL-SUBSTITUTION-STATEMENTS:   */
          /*-----------------------------------------------*/

          CALL LINEFD(LST_PRF_SBST_STMT);
DO        I=MAX(LST_PRF_BLCK_STMT,LST_SUF_BLCK_STMT)+1 TO
            LST_PRF_SBST_STMT;
```

```
          /*----------------------------------------*/
          /* IDENTIFICATION OF A SEQUENCE OF STATE- */
          /* MENTS WITH EQUAL SUBSTITUENDA  A N D   */
          /* SUBSTITUENTIA AND - OPTIONALLY - COMPU- */
          /* TATION OF THE ABSOLUTE AND RELATIVE PRE- */
          /* QUENCIES OF THE SOURCES IN THE LEARNING- */
          /* CORPUS TO WHICH THESE STATEMENTS ARE AP- */
          /* PLICABLE:                              */
          /*----------------------------------------*/

        IF STATISTICS THEN FREQ_SRC_EQ_XY=FREQ_SRC(I);
        X = SUBSTR(X_Z(I),1,LGT_X(I));
DO      J=I+1 TO LST_PRF_SBST_STMT
           WHILE(X=SUBSTR(X_Z(J),1,LGT_X(J)) & Y(I)=Y(J));
        IF STATISTICS
         THEN FREQ_SRC_EQ_XY=FREQ_SRC_EQ_XY+FREQ_SRC(J);
END;
        J=J-1;

          /*----------------------------------------*/
          /* FOR THE IDENTIFIED SEQUENCE OF STATE-  */
          /* MENTS PREPARATION OF THE SUBSTITUENS AND */
          /* THE SUBSTITUENDUM AND - OPTICNALLY - OF */
          /* THE FREQUENCIES FOR THE OUTPUT:        */
          /*----------------------------------------*/

        IF STATISTICS
         THEN PUT STRING(WRKCHR) EDIT
              (100.000*FREQ_SRC_EQ_XY/RECNO_CORPUS,'% | ',
              FREQ_SRC_EQ_XY,' | ''',X,''' -> ')
              (F(6,2),A,F(6),3 A);
         ELSE WRKCHR=''''||X||''' -> ';
        IF LENGTH(WRKCHR)+LENGTH(Y(J))+3>60
         THEN CALL OUTINIT;
        WRKCHR = WRKCHR||''''||Y(J)||'''/';

          /*----------------------------------------*/
          /* FOR THE IDENTIFIED SEQUENCE OF STATE-  */
          /* MENTS PREPARATION OF THE CONTEXTUAL-   */
          /* STRINGS FOR THE OUTPUT:                */
          /*----------------------------------------*/

        IF LENGTH(WRKCHR)+LENGTH(X_Z(J))-LGT_X(J)+3+SIGN(
           LENGTH(X_Z(J))-LGT_X(J))*(1+(SIGN(I-J)+1)*1)>60
         THEN CALL OUTINIT;
```

```
          IF I=J
 THEN     IF LGT_X(J)=LENGTH(X_Z(J))
           THEN PUT SKIP EDIT(WRKCHR,'#__') (CCL(11),2 A);
           ELSE IF SUBSTR(X_Z(J),LENGTH(X_Z(J)))=' '
            THEN IF LENGTH(X_Z(J))-LGT_X(J)=1
             THEN PUT SKIP EDIT(WRKCHR,'#__#') (CCL(11),2 A);
             ELSE PUT SKIP EDIT(WRKCHR,'#__''',SUBSTR(X_Z(J),
                  LGT_X(J)+1,LENGTH(X_Z(J))-LGT_X(J)-1),
                  '''#')
                  (COL(11),4 A);
            ELSE PUT SKIP EDIT(
                 WRKCHR,'#__''',SUBSTR(X_Z(J),LGT_X(J)+1),
                 '''')
                 (COL(11),4 A);
 ELSE DO;
          WRKCHR = WRKCHR||'#__(';
  DO      K=I TO J;
          IF SUBSTR(X_Z(K),LENGTH(X_Z(K)))=' '
           THEN IF LENGTH(X_Z(K))-LGT_X(K)=1
            THEN Z='#|';
            ELSE Z='''||SUBSTR(X_Z(K),LGT_X(K)+1,
                 LENGTH(X_Z(K))-LGT_X(K)-1)||'''#|';
           ELSE Z='''||SUBSTR(X_Z(K),LGT_X(K)+1)||'''|';
          IF LENGTH(WRKCHR)+LENGTH(Z)>60 THEN CALL OUTINIT;
          WRKCHR = WRKCHR||Z;
  END;
          SUBSTR(WRKCHR,LENGTH(WRKCHR))=')';
          CALL OUTINIT;
 END;
          STMT_COUNTER = STMT_COUNTER+1;
          I            = J;
END;

          /*-----------------------------------------------*/
          /* OUTPUT OF SUFFIXAL-SUBSTITUTION-STATEMENTS:   */
          /*-----------------------------------------------*/

          CALL LINEFD(LST_SUF_SBST_STMT);
DO        I=LST_PRF_SBST_STMT+1 TO LST_SUF_SBST_STMT;
```

```
        /*----------------------------------------*/
        /* IDENTIFICATION OF A SEQUENCE OF STATE- */
        /* MENTS WITH EQUAL SUBSTITUENDA  A N D   */
        /* SUBSTITUENTIA AND - OPTIONALLY - COMPU- */
        /* TATION OF THE ABSOLUTE AND RELATIVE FRE- */
        /* QUENCIES OF THE SOURCES IN THE LEARNING- */
        /* CORPUS TO WHICH THESE STATEMENTS ARE AP- */
        /* PLICABLE:                              */
        /*----------------------------------------*/

      IF STATISTICS THEN FREQ_SRC_EQ_XY=FREQ_SRC(I);
      X = SUBSTR(X_Z(I),1,LGT_X(I));
DO    J=I+1 TO LST_SUF_SBST_STMT
        WHILE(X=SUBSTR(X_Z(J),1,LGT_X(J)) & Y(I)=Y(J));
      IF STATISTICS
       THEN FREQ_SRC_EQ_XY=FREQ_SRC_EQ_XY+FREQ_SRC(J);
END;
      J=J-1;

        /*----------------------------------------*/
        /* FOR THE IDENTIFIED SEQUENCE OF STATE-  */
        /* MENTS PREPARATION OF THE SUBSTITUENS AND */
        /* THE SUBSTITUENDUM AND - OPTIONALLY - OF */
        /* THE FREQUENCIES FOR THE OUTPUT:        */
        /*----------------------------------------*/

      IF STATISTICS
       THEN PUT STRING(WRKCHR) EDIT
            (100.000*FREQ_SRC_EQ_XY/RECNO_CORPUS,'% | ',
            FREQ_SRC_EQ_XY,' | ''',INVERT(X,0),''' -> ')
            (F(6,2),A,F(6),3 A);
       ELSE WRKCHR=''''||INVERT(X,0)||''' -> ';
      IF LENGTH(WRKCHR)+LENGTH(Y(J))+3>60
       THEN CALL OUTINIT;
      WRKCHR = WRKCHR||''''||INVERT(Y(J),0)||'''/';

        /*----------------------------------------*/
        /* FOR THE IDENTIFIED SEQUENCE OF STATE-  */
        /* MENTS PREPARATION OF THE CONTEXTUAL-   */
        /* STRINGS FOR THE OUTPUT:                */
        /*----------------------------------------*/

      IF LENGTH(WRKCHR)+LENGTH(X_Z(J))-LGT_X(J)+3+SIGN(
         LENGTH(X_Z(J))-LGT_X(J))*(1+(SIGN(I-J)+1)*1)>60
       THEN CALL OUTINIT;
```

```
          IF I=J
 THEN     IF LGT_X(J)=LENGTH(X_Z(J))
           THEN PUT SKIP EDIT(WRKCHR,'__#') (COL(11),2 A);
           ELSE IF SUBSTR(X_Z(J),LENGTH(X_Z(J)))=' '
            THEN PUT SKIP EDIT(
                 WRKCHR,'#''',INVERT(SUBSTR(X_Z(J),LGT_X(J)+1
                 ,LENGTH(X_Z(J))-LGT_X(J)-1),0),'''__#')
                 (COL(11),4 A);
            ELSE PUT SKIP EDIT(
                 WRKCHR,'''',INVERT(SUBSTR(X_Z(J),LGT_X(J)+1)
                 ,0),'''__#')
                 (COL(11),4 A);
 ELSE DO;
          WRKCHR = WRKCHR||'(';
  DO      K=J TO I BY -1;
          IF SUBSTR(X_Z(K),LENGTH(X_Z(K)))=' '
           THEN Z='#'''||INVERT(SUBSTR(X_Z(K),LGT_X(K)+1,
                  LENGTH(X_Z(K))-LGT_X(K)-1),0)||'''|';
           ELSE Z=''''||INVERT(SUBSTR(X_Z(K),LGT_X(K)+1),0)||
                  '''|';
          IF LENGTH(WRKCHR)+LENGTH(Z)>60 THEN CALL OUTINIT;
          WRKCHR = WRKCHR||Z;
  END;
          SUBSTR(WRKCHR,LENGTH(WRKCHR))=')';
          IF LENGTH(WRKCHR||'__#')>60
           THEN PUT SKIP EDIT(WRKCHR,INDENT,'__#')
                (COL(11),A,COL(11),2 A);
           ELSE PUT SKIP EDIT(WRKCHR,'__#')
                (COL(11),2 A);
 END;
          STMT_COUNTER = STMT_COUNTER+1;
          I            = J;
END;

/*-------------------------------------------------------*/
/* OUTPUT OF THE NUMBER OF THE STATEMENTS AND OF THE NUM- */
/* BER OF SOURCE-TARGET-PAIRS FROM WHICH THE STATEMENTS   */
/* WERE INFERED:                                          */
/*-------------------------------------------------------*/

          REVERT ENDPAGE(SYSPRINT);
          IF LINENO(SYSPRINT)>55
           THEN SIGNAL ENDPAGE(SYSPRINT);
           ELSE PUT SKIP(2);
```

```
         PUT SKIP EDIT(
         'NUMBER OF SOURCE-TARGET-PAIRS',
         'IN THE LEARNING-CORPUS:          ',RECNO_CORPUS,
         'NUMBER OF GENERATED STATEMENTS:  ',STMT_COUNTER)
         (2 (COL(11),A),F(6),COL(11),A,F(5));

/*-----------------------------------------------------------*/
/* SUBROUTINE-PROCEDURE FOR THE OUTPUT AND INITIALIZATION    */
/* OF WRKCHR:                                                */
/*-----------------------------------------------------------*/

OUTINIT: PROC;
         PUT SKIP EDIT(WRKCHR) (COL(11),A);
         WRKCHR = INDENT;
END      OUTINIT;

/*-----------------------------------------------------------*/
/* SUBROUTINE-PROCEDURE FOR THE SKIPPING OF A LINE BEFORE    */
/* THE FIRST STATEMENT OF A STATEMENT-TYPE IS WRITTEN ON-    */
/* TO THE SYSPRINT-FILE:                                     */
/*-----------------------------------------------------------*/

LINEFD:  PROC(LST_STMT);
DCL      LST_STMT              BIN FIXED(15);

         IF STMT_COUNTER>0 & LST_STMT>0
          THEN IF LINENO(SYSPRINT)<59
THEN DO;
         IF STATISTICS
          THEN PUT SKIP EDIT
               ('|','|') (COL(19),A,COL(28),A);
          ELSE PUT SKIP;
END;
ELSE      SIGNAL ENDPAGE(SYSPRINT);

END      LINEFD;

END      SIMTOUT;
```

```
/*-------------------------------------------------------------*/
/* NAME: TODSET                                                */
/*                                                             */
/* FUNCTION:                                                   */
/* TODSET IS A SUBROUTINE-PROCEDURE. IT STORES A SET OF        */
/* DERIVATION-STATEMENTS IN STMTFLE. TODSET STORES EACH        */
/* SET OF DERIVATION-STATEMENTS IN 2 WAYS IN STMTFLE:          */
/*  - AT FIRST THE UNPREPARED DERIVATION-STATEMENTS, I.        */
/*    E. THE DERIVATION-STATEMENTS AS THEY ARE PRODUCED        */
/*    BY DISCOV, ARE WRITTEN INTO STMTFLE FOR THEIR FUR-       */
/*    THER PROCESSING BY STMTOUT.                              */
/*  - SECONDLY THE DERIVATION-STATEMENTS ARE PREPARED FOR      */
/*    THEIR LATER APPLICATION BY DERIVE AND THESE PRE-         */
/*    PARED STATEMENTS ARE WRITTEN INTO STMTFLE.               */
/* IN DETAIL STMTFLE HAS THE FOLLOWING SETUP AFTER IT HAS      */
/* BEEN FILLED BY TODSET:                                      */
/*  - THE FIRST RECORD CONTAINS CONTROL-DATA ABOUT THE         */
/*    SET OF DERIVATION-STATEMENTS. THESE CONTROL-DATA         */
/*    ARE:                                                     */
/*      - THE POWER OF THE LEARNING-CORPUS FROM WHICH THE      */
/*        SET OF DERIVATION-STATEMENTS WAS INFERED (=          */
/*        RECNO_CORPUS),                                       */
/*      - THE ADDRESS OF THE LAST PREFIXAL-BLOCKING-STATE-     */
/*        MENT (= LST_PRF_BLCK_STMT),                          */
/*      - THE ADDRESS OF THE LAST SUFFIXAL-BLOCKING-STATE-     */
/*        MENT (= LST_SUF_BLCK_STMT),                          */
/*      - THE ADDRESS OF THE LAST PREFIXAL-SUBSTITUTION-       */
/*        STATEMENT (= LST_PRF_SBST_STMT),                     */
/*      - THE ADDRESS OF THE LAST SUFFIXAL-SUBSTITUTION-       */
/*        STATEMENT (= LST_SUF_SBST_STMT).                     */
/*  - THE FOLLOWING N RECORDS (N := MAX(LST_PRF_BLCK_STMT,     */
/*    LST_SUF_BLCK_STMT, LST_PRF_SBST_STMT, LST_SUF_SBST_      */
/*    STMT)) CONTAIN THE UNPREPARED SET OF DERIVATION-         */
/*    STATEMENTS INTENDED FOR THE FURTHER PROCESSING BY        */
/*    STMTOUT.                                                 */
/*  - THE FOLLOWING N RECORDS (FOR A DEFINITION OF N, S.       */
/*    A.) CONTAIN THE SET OF DERIVATION-STATEMENTS PRE-        */
/*    PARED FOR DERIVE.                                        */
/* FINALLY TODSET WRITES AN INFORMATORY MESSAGE ONTO THE       */
/* SYSPRINT-FILE, WHICH INDICATES THAT THE SET OF DERIVA-      */
/* TION-STATEMENTS WAS STORED IN STMTFLE.                      */
/*                                                             */
/* FILE:                                                       */
/*  OUTPUT: STMTFLE.                                           */
/*-------------------------------------------------------------*/

TODSET:  PROC REORDER;
DCL      1 STMT(*)                  CTL EXT,
          2 X_Z                     CHAR(30) VAR,
          2 LGT_X                   BIN FIXED(15),
          2 Y                       CHAR(30) VAR,
          2 FREQ_SRC_OR_LLCXZ       BIN FIXED(31),
```

```
        1 BOOKEEP                     EXT,
         2 RECNO_CORPUS               BIN FIXED(31),
         2 LST_PRF_BLCK_STMT          BIN FIXED(15),
         2 LST_SUF_BLCK_STMT          BIN FIXED(15),
         2 LST_PRF_SBST_STMT          BIN FIXED(15),
         2 LST_SUF_SBST_STMT          BIN FIXED(15),
         2 DUMMY                      CHAR(58) INIT(' '),

        MAXSTNO                       BIN FIXED(15) EXT,
        I                             BIN FIXED(15),
        (LENGTH,MAX)                  BUILTIN,
        STMTFLE                       FILE RECORD SEQL,
        SYSPRINT                      FILE STREAM PRINT;

        OPEN FILE(STMTFLE) OUTPUT;

/*-------------------------------------------------------*/
/* OUTPUT OF THE CONTROL-DATA:                           */
/*-------------------------------------------------------*/

        WRITE FILE(STMTFLE) FROM(BOOKEEP);

/*-------------------------------------------------------*/
/* OUTPUT OF THE UNPREPARED DERIVATION-STATEMENTS FOR THE */
/* FURTHER PROCESSING BY STMTOUT:                        */
/*-------------------------------------------------------*/

DO      I=1 TO MAX(LST_PRF_BLCK_STMT,LST_SUF_BLCK_STMT,
                   LST_PRF_SBST_STMT,LST_SUF_SBST_STMT);
        WRITE FILE(STMTFLE) FROM(STMT(I));
END;

/*-------------------------------------------------------*/
/* PREPARATION AND OUTPUT OF THE DERIVATION-STATEMENTS    */
/* FOR THE FURTHER PROCESSING BY DERIVE:                 */
/*-------------------------------------------------------*/
        /*-----------------------------------------------*/
        /* PREPARATION AND OUTPUT OF BLOCKING-STATE-     */
        /* MENTS:                                        */
        /*-----------------------------------------------*/

DO      I=1 TO MAX(LST_PRF_BLCK_STMT,LST_SUF_BLCK_STMT);
        FREQ_SRC_OR_LLCXZ(I)=LENGTH(X_Z(I));
        WRITE FILE(STMTFLE) FROM(STMT(I));
END;
```

```
        /*-------------------------------------------------*/
        /* PREPARATION AND OUTPUT OF PREFIXAL-SUBSTITU-    */
        /* TION-STATEMENTS:                                */
        /*-------------------------------------------------*/

        CALL PREPOUT(MAX(LST_PRF_BLCK_STMT,
                         LST_SUF_BLCK_STMT)+1,
                     LST_PRF_SBST_STMT);

        /*-------------------------------------------------*/
        /* PREPARATION AND OUTPUT OF SUFFIXAL-SUBSTITU-    */
        /* TION-STATEMENTS:                                */
        /*-------------------------------------------------*/

        CALL PREPOUT(LST_PRF_SBST_STMT+1,
                     LST_SUF_SBST_STMT);

        CLOSE FILE(STMTFLE);

/*-----------------------------------------------------------*/
/* FINAL OUTPUT OF AN INFORMATORY MESSAGE ONTO THE           */
/* SYSPRINT-FILE:                                            */
/*-----------------------------------------------------------*/

        SIGNAL ENDPAGE(SYSPRINT);
        PUT SKIP EDIT
        ('INFORMATORY MESSAGE:')                  (COL(11),A)
        ((20)'-')                                 (COL(11),A)
        ('THE GENERATED DERIVATION-STATEMENTS',
        ' WERE STORED IN THE DATA ') (SKIP(2),COL(11),2 A)
        ('SET ALLOCATED TO FILE STMTFLE.')        (COL(11),A);
```

```
/*-------------------------------------------------------------*/
/* SUBROUTINE-PROCEDURE FOR THE PREPARATION AND OUTPUT OF      */
/* DERIVATION-STATEMENTS FOR THE FURTHER PROCESSING BY         */
/* DERIVE. AT EACH INVOCATION PREPOUT PROCESSES ALL DERI-      */
/* VATION-STATEMENTS WITH A SUBSCRIPT GREATER THAN OR          */
/* EQUAL TO START AND LESS THAN OR EQUAL TO LIMIT. PREP-       */
/* ARATION AND OUTPUT CONSIST OF THE FOLLOWING STEPS:          */
/*  - A BINARY SORT-TREE IS CREATED FOR THE SORTING OF         */
/*    THE DERIVATION-STATEMENTS IN ASCENDING ORDER. (THE       */
/*    KEY WHICH GOVERNS THE SORTING IS THE STRING X_Z(I)       */
/*    OF EACH DERIVATION-STATEMENT STMT(I) (START<=I<=         */
/*    LIMIT).)                                                 */
/*  - THE BINARY TREE IS LINEARIZED.                           */
/*  - FOR EACH STRING X_Z(J) (START<=J<=LIMIT) THE LENGTH      */
/*    OF THE LONGEST STRING X_Z(K) (START<=K<=LIMIT) OF        */
/*    WHICH X_Z(J) IS A LEFT SUBSTRING IS COMPUTED AND         */
/*    STORED WITH X_Z(J). THE LINEARIZED TREE IS WRITTEN       */
/*    ONTO THE STMTFLE-FILE.                                   */
/* THE ALGORITHM FOR THE CREATION AND LINEARIZATION OF         */
/* THE TREE HAS BEEN TAKEN FROM: BRAUN, S. (1974): ALGO-       */
/* RITHMISCHE LINGUISTIK. STUTTGART/BERLIN/KOELN/MAINZ.        */
/* P. 52-55.                                                   */
/*-------------------------------------------------------------*/

PREPOUT: PROC(START,LIMIT);
DCL      POINT(0:MAXSTNO,2)  BIN FIXED(15) INIT((2*START)*,
                             (2*(LIMIT-START+1))0),
         START               BIN FIXED(15),
         LIMIT               BIN FIXED(15),
         (J,K)               BIN FIXED(15),
         (MIN,SUBSTR)        BUILTIN;

         IF LIMIT=0 THEN RETURN;

         /*-----------------------------------------------*/
         /* CREATION OF THE BINARY SORT-TREE:             */
         /*-----------------------------------------------*/

         POINT(0,2) = START;
DO       I=START+1 TO LIMIT;
         J          = 0;
         K          = 2;
 DO      UNTIL(POINT(J,K)=0);
         J          = POINT(J,K);
         IF X_Z(I)<=X_Z(J)
          THEN K=1;
          ELSE K=2;
 END;
         POINT(J,K) = I;
END;
```

```
        /*-------------------------------------------------*/
        /* LINEARIZATION OF THE TREE:                      */
        /*-------------------------------------------------*/

        J           = 0;
        POINT(0,1) = 0;
DO      WHILE(POINT(J,1)¬=0 | POINT(J,2)¬=0);
        IF POINT(J,1)=0
 THEN   I           = J;
 ELSE DO;
        K           = POINT(J,1);
  DO    WHILE(POINT(K,2)>0);
        K           = POINT(K,2);
  END;
        POINT(K,2) = POINT(I,2);
        POINT(I,2) = POINT(J,1);
        POINT(J,1) = 0;
 END;
        J           = POINT(I,2);
END;

        /*-------------------------------------------------*/
        /* FOR EACH X_Z(J) (START<=J<=LIMIT) COMPUTATION */
        /* OF THE LENGTH OF THE LONGEST X_Z(K) (START<=K */
        /* <=LIMIT) OF WHICH X_Z(J) IS A LEFT SUBSTRING; */
        /* OUTPUT OF THE LINEARIZED-TREE:                  */
        /*-------------------------------------------------*/

DO      J=POINT(0,2) REPEAT POINT(J,2) WHILE(J¬=0);
        FREQ_SRC_OR_LLCXZ(J) = LENGTH(X_Z(J));
 DO     K=POINT(J,2) REPEAT POINT(K,2)
           WHILE(K¬=0 & X_Z(J)=SUBSTR(X_Z(K),1,
                 MIN(LENGTH(X_Z(J)),LENGTH(X_Z(K)))));
        IF LENGTH(X_Z(K))>FREQ_SRC_OR_LLCXZ(J)
         THEN FREQ_SRC_OR_LLCXZ(J)=LENGTH(X_Z(K));
 END;
        WRITE FILE(STMTFLE) FROM(STMT(J));
END;

END     PREPOUT;

END     TODSET;
```

```
/*-------------------------------------------------------------*/
/* NAME: FRODSET                                               */
/*                                                             */
/* FUNCTION:                                                   */
/* FRODSET IS A SUBROUTINE-PROCEDURE. IT READS A SET OF        */
/* DERIVATION-STATEMENTS AND THE CONTROL-DATA ABOUT THAT       */
/* SET OF DERIVATION-STATEMENTS FROM STMTFLE:                  */
/*  - AT FIRST FRODSET READS THE FIRST RECORD OF STMTFLE       */
/*    INTO THE STRUCTURE BOOKEEP. THIS RECORD CONTAINS         */
/*    THE FOLLOWING CONTROL-DATA ABOUT THE 2 SETS OF DE-       */
/*    RIVATION-STATEMENTS STORED IN THE SUCCEEDING REC-        */
/*    ORDS OF STMTFLE:                                         */
/*     - THE POWER OF THE LEARNING-CORPUS FROM WHICH THE       */
/*       DERIVATION-STATEMENTS WERE INFERED,                   */
/*     - THE ADDRESS OF THE LAST PREFIXAL-BLOCKING-STATE-      */
/*       MENT,                                                 */
/*     - THE ADDRESS OF THE LAST SUFFIXAL-BLOCKING-STATE-      */
/*       MENT,                                                 */
/*     - THE ADDRESS OF THE LAST PREFIXAL-SUBSTITUTION-        */
/*       STATEMENT,                                            */
/*     - THE ADDRESS OF THE LAST SUFFIXAL-SUBSTITUTION-        */
/*       STATEMENT.                                            */
/*  - THEN FRODSET READS INTO THE STRUCTURE-ARRAY STMT -       */
/*    DEPENDENT ON THE VALUE OF ITS PARAMETER MODE -           */
/*     - E I T H E R  THE FOLLOWING N RECORDS OF STMTFLE       */
/*       (N := MAX(LST_PRF_BLCK_STMT,LST_SUF_BLCK_STMT,        */
/*       LST_PRF_SEST_STMT,LST_SUF_SEST_STMT)), WHICH          */
/*       CONTAIN THE SET OF DERIVATION-STATEMENTS PRE-         */
/*       PARED FOR THE FURTHER PROCESSING BY STMTOUT.          */
/*     - O R  THE LAST N RECORDS OF STMTFLE (FOR A DEFI-       */
/*       NITION OF N, S. A.), WHICH CONTAIN THE SET OF         */
/*       DERIVATION-STATEMENTS PREPARED FOR THE FURTHER        */
/*       PROCESSING BY DERIVE.                                 */
/* IF STMTFLE IS EMPTY OR IF STMTFLE CONTAINS MORE DERI-       */
/* VATION-STATEMENTS THAN PRISM CAN PROCESS, FRODSET           */
/* WRITES AN ERROR-MESSAGE ONTO THE SYSPRINT-FILE, AND IT      */
/* ASSIGNS TO ITS PARAMETER RETURN_CODE THE VALUE 1.           */
/*                                                             */
/* PARAMETERS;                                                 */
/*  MODE:         CHAR(11) VAR.                                */
/*                - IF MODE='PRINT_STMTS', FRODSET READS       */
/*                  INTO STMT THE SET OF DERIVATION-STATE-     */
/*                  MENTS INTENDED FOR STMTOUT.                */
/*                - IF MODE='APPLY', FRODSET READS INTO        */
/*                  STMT THE SET OF DERIVATION-STATEMENTS      */
/*                  INTENDED FOR DERIVE.                       */
/*  RETURN_CODE: BIN FIXED(31). IF STMTFLE IS EMPTY OR IF      */
/*               STMTFLE CONTAINS MORE DERIVATION-STATE-       */
/*               MENTS THAN PRISM CAN PROCESS, FRODSET         */
/*               ASSIGNS TO RETURN_CODE THE VALUE 1.           */
/*-------------------------------------------------------------*/
```

```
/*-------------------------------------------------------*/
/* FILE:                                                 */
/*  INPUT: STMTFLE.                                      */
/*-------------------------------------------------------*/

PRODSET: PROC(MODE,RETURN_CODE) REORDER;
DCL      1 STMT(*)              CTL EXT,
          2 X_Z                 CHAR(30) VAR,
          2 LGT_X               BIN FIXED(15),
          2 Y                   CHAR(30) VAR,
          2 FREQ_SRC_OR_LLCXZ   BIN FIXED(31),

         1 BOOKEEP              EXT,
          2 RECNO_CORPUS        BIN FIXED(31),
          2 LST_PRF_BLCK_STMT   BIN FIXED(15),
          2 LST_SUF_BLCK_STMT   BIN FIXED(15),
          2 LST_PRF_SBST_STMT   BIN FIXED(15),
          2 LST_SUF_SBST_STMT   BIN FIXED(15),
          2 DUMMY               CHAR(58),

         MODE                   CHAR(11) VAR,
         MAXSTNO                BIN FIXED(15) EXT,
         STMT_NO                BIN FIXED(15),
         I                      BIN FIXED(15),
         RETURN_CODE            BIN FIXED(31),
         MAX                    BUILTIN,
         STMTFLE                FILE RECORD SEQI,
         SYSPRINT               FILE STREAM PRINT;

/*-------------------------------------------------------*/
/* OUTPUT OF AN ERROR-MESSAGE ONTO THE SYSPRINT-FILE IF  */
/* STMTFLE IS EMPTY:                                     */
/*-------------------------------------------------------*/

ON       ENDFILE(STMTFLE) BEGIN;
         SIGNAL ENDPAGE(SYSPRINT);
         PUT SKIP EDIT(
         '*** ERROR WHILE TRYING TO READ DERIVATION-STATEM',
         'ENTS FROM',
         '    THE DATA SET ALLOCATED TO FILE STMTFLE AND ',
         'BREAK OF',
         '    PRISM: THE DATA SET WAS EMPTY.')
         (COL(11),2 A);
         RETURN_CODE = 1;
         GOTO EXIT;
END;

         OPEN FILE(STMTFLE) INPUT;
```

```
/*-------------------------------------------------------*/
/* INPUT OF THE CONTROL-DATA:                            */
/*-------------------------------------------------------*/

          READ FILE(STMTFLE) INTO(BOOKEEP);

          STMT_NO = MAX(LST_PRF_BLCK_STMT,LST_SUF_BLCK_STMT,
                        LST_PRF_SBST_STMT,LST_SUF_SBST_STMT);

/*-------------------------------------------------------*/
/* OUTPUT OF AN ERROR-MESSAGE ONTO THE SYSPRINT-FILE IF  */
/* STMTFLE CONTAINS MORE DERIVATION-STATEMENTS THAN PRISM */
/* CAN PROCESS:                                          */
/*-------------------------------------------------------*/

          IF STMT_NO>MAXSTNO
THEN DO;
          SIGNAL ENDPAGE(SYSPRINT);
          PUT SKIP EDIT(
          '*** ERROR WHILE TRYING TO READ THE DERIVATION-',
          'STATEMENTS',
          '    FROM THE DATA SET ALLOCATED TO FILE STMTFLE ',
          'AND BREAK OF',
          '    PRISM: THE DATA SET CONTAINS MORE DERIVATION',
          '-STATEMENTS',
          '    THAN PRISM PROCESS.')
          (COL(11),2 A);
          RETURN_CODE = 1;
          GOTO EXIT;
END;

SELECT(MODE);

/*-------------------------------------------------------*/
/* INPUT OF THE DERIVATION-STATEMENTS FOR STMTOUT:       */
/*-------------------------------------------------------*/

 WHEN('PRINT_STMTS') DO I=1 TO STMT_NO;
          READ FILE(STMTFLE) INTO(STMT(I));
 END;

/*-------------------------------------------------------*/
/* INPUT OF THE DERIVATION-STATEMENTS FOR DERIVE:        */
/*-------------------------------------------------------*/

 WHEN('APPLY') DO;
          READ FILE(STMTFLE) IGNORE(STMT_NO);
  DO      I=1 TO STMT_NO;
          READ FILE(STMTFLE) INTO(STMT(I));
  END;
 END;
END;
```

```
          CLOSE FILE(STMTFIE);
EXIT:     END PRODSET;
```

```
/*-------------------------------------------------------*/
/* NAME: APPLY                                           */
/*                                                       */
/* FUNCTION:                                             */
/* APPLY IS A SUBROUTINE-PROCEDURE. IT READS SOURCES FROM */
/* CORPUS AND ACTIVATES THE FUNCTION-PROCEDURE DERIVE     */
/* WITH THESE SOURCES AS ARGUMENTS. DERIVE THEN RETURNS   */
/* FOR EACH SOURCE                                       */
/*  - E I T H E R  A TARGET, WHICH IT HAS GENERATED BY    */
/*    THE APPLICATION OF A GIVEN SET OF DERIVATION-STATE- */
/*    MENTS TO THE SOURCE.                               */
/*  - O R  A MESSAGE, THAT INFORMS ABOUT THE REASON WHY A */
/*    TARGET CANNOT BE GENERATED.                        */
/* EACH SOURCE AND ITS GENERATED TARGET OR MESSAGE IS     */
/* WRITTEN BACK TO CORPUS BY APPLY. FINALLY APPLY WRITES  */
/* AN INFORMATORY MESSAGE ONTO THE SYPRINT-FILE, WHICH    */
/* INDICATES THE CORRECT EXECUTION OF APPLY.              */
/*                                                       */
/* FILE:                                                 */
/*  UPDATE: CORPUS.                                      */
/*-------------------------------------------------------*/

APPLY:   PROC REORDER;
DCL      REC                CHAR(80),
         SOURCE             CHAR(30) DEF REC POS(11),
         TARGET             CHAR(30) DEF REC POS(41),
         CORPUS             FILE RECORD SEQL,
         SYSPRINT           FILE STREAM PRINT,
         DERIVE             ENTRY(CHAR(30) VAR)
                               RETURNS(CHAR(30) VAR);

ON       ENDFILE(CORPUS) GOTO SHUT;
         OPEN FILE(CORPUS) UPDATE;

DO       WHILE('1'B);
         READ FILE(CORPUS) INTO(REC);
         TARGET = DERIVE(SOURCE);
         REWRITE FILE(CORPUS) FROM(REC);
END;

SHUT:    CLOSE FILE(CORPUS);
         SIGNAL ENDPAGE(SYSPRINT);
```

```
        PUT SKIP EDIT
        ('INFORMATORY MESSAGE:') (COL(11),A)
        ((20)'-')                (COL(11),A)
        ('FOR EACH SOURCE OF THE DATA SET ALLOCATED TO ',
        'FILE CORPUS EI-',
        'THER A TARGET WAS GENERATED OR - IF NO TARGET ',
        'COULD BE GEN-',
        'ERATED - A MESSAGE WAS ISSUED. EACH SOURCE WAS ',
        'STORED WITH',
        'ITS TARGET OR MESSAGE IN THE DATA SET ALLOCATED ',
        'TO FILE COR-',
        'PUS.')
        (SKIP(2),5 (COL(11),2 A));

END     APPLY;
```

```
/*-------------------------------------------------------------*/
/* NAME: DERIVE                                                */
/*                                                             */
/* FUNCTION:                                                   */
/* DERIVE IS A FUNCTION-PROCEDURE. IT RETURNS                  */
/*  - E I T H E R  A TARGET WHICH RESULTS FROM THE APPLI-      */
/*    CATION OF DERIVATION-STATEMENTS TO ITS ARGUMENT          */
/*    SOURCE.                                                  */
/*  - O R  A MESSAGE, THAT INFORMS ABOUT THE REASON WHY A      */
/*    TARGET CANNOT BE GENERATED.                              */
/*                                                             */
/* PARAMETER:                                                  */
/*  SOURCE: CHAR(30) VAR. A STRING WHICH CONTAINS - LEFT       */
/*          ADJUSTED - A SOURCE-STRING, FOR WHICH A TAR-       */
/*          GET SHALL BE GENERATED. (IF THERE IS A RIGHT       */
/*          SUBSTRING OF THE VARIABLE SOURCE WHICH DOES        */
/*          NOT BELONG TO THE SOURCE-STRING, THIS RIGHT        */
/*          SUBSTRING MUST BE FILLED WITH BLANKS.              */
/*                                                             */
/* RETURNED VALUE:                                             */
/*  CHAR(30) VAR. DETAILED DESCRIPTION: S. A.                  */
/*-------------------------------------------------------------*/

DERIVE:  PROC(SOURCE) RETURNS(CHAR(30) VAR) RECURSIVE;
DCL      1 STMT(*)              CTL EXT,
          2 X_Z                 CHAR(30) VAR,
          2 LGT_X               BIN FIXED(15),
          2 Y                   CHAR(30) VAR,
          2 LGT_LGST_CNTNG_XZ   BIN FIXED(31),

         1 BOOKEEP              EXT,
          2 RECNO_CORPUS        BIN FIXED(31),
          2 LST_PRF_BLCK_STMT   BIN FIXED(15),
          2 LST_SUF_BLCK_STMT   BIN FIXED(15),
          2 LST_PRF_SBST_STMT   BIN FIXED(15),
          2 LST_SUF_SBST_STMT   BIN FIXED(15),
          2 DUMMY               CHAR(58),

         SOURCE                 CHAR(30) VAR,
         ISOURCE                CHAR(30) VAR,
         TARGET                 CHAR(30) VAR,
         NO_APPL_PS_STMT        BIN FIXED(15),
         NO_APPL_SS_STMT        BIN FIXED(15),
         (HIGH,INDEX,LENGTH)    BUILTIN,
         (LOW,MAX,SUBSTR)       BUILTIN,
         VERIFY                 BUILTIN,
         INVERT                 ENTRY(CHAR(*),BIN FIXED(15))
                                   RETURNS(CHAR(30) VAR);
```

```
/*-------------------------------------------------------*/
/* SYNTAX-CHECKING OF THE SOURCE-FIELD AND - IF THE      */
/* SOURCE-FIELD CONTAINS AN ERROR - OUTPUT OF AN ERROR-  */
/* MESSAGE:                                              */
/*-------------------------------------------------------*/

         IF LENGTH(SOURCE)=0
          | INDEX(SOURCE,' ')=1
          | VERIFY(SUBSTR(SOURCE||' ',INDEX(SOURCE||' ',
                 ' '))),' ')>0
          | INDEX(SOURCE,LOW(1))>0
          | INDEX(SOURCE,HIGH(1))>0
          THEN TARGET = ' *** ERROR IN SOURCE-FIELD.';
          ELSE
DO;
         SOURCE  = SUBSTR(SOURCE,1,INDEX(SOURCE||' ',' ')
                        -1);
         ISOURCE = INVERT(SOURCE,0);

/*-------------------------------------------------------*/
/* TEST FOR THE APPLICABILITY OF A BLOCKING-STATEMENT AND */
/* - EVENTUALLY - APPLICATION OF A BLOCKING-STATEMENT:   */
/*-------------------------------------------------------*/

         IF ADAPSTM(SOURCE,1,LST_PRF_BLCK_STMT) >0
          | ADAPSTM(ISOURCE,LST_PRF_BLCK_STMT+1,
                          LST_SUF_BLCK_STMT)  >0
 THEN    TARGET = '';

/*-------------------------------------------------------*/
/* TEST FOR THE APPLICABILITY OF SUBSTITUTION-STATEMENTS  */
/* AND                                                   */
/*  - E I T H E R  ISSUING OF A MESSAGE, THAT INFORMS    */
/*    ABOUT THE REASON WHY A TARGET CANNOT BE GENERATED, */
/*  - O R  APPLICATION OF A PREFIXAL- AND A SUFFIXAL-SUB- */
/*    STITUTION-STATEMENT:                               */
/*-------------------------------------------------------*/

 ELSE DO;
         NC_APPL_PS_STMT = ADAPSTM(SOURCE,
                                   MAX(LST_PRF_BLCK_STMT,
                                       LST_SUF_BLCK_STMT)+1,
                                   LST_PRF_SBST_STMT);
         NC_APPL_SS_STMT = ADAPSTM(ISOURCE,LST_PRF_SBST_STMT
                                   +1,LST_SUF_SBST_STMT);
  SELECT;
```

```
   WHEN(NC_APPL_PS_STMT=0 | NO_APPL_SS_STMT=0)
        TARGET = ' *** STMTS CANNOT BE APPLIED.';
   WHEN(LENGTH(SOURCE)-LGT_X(NO_APPL_SS_STMT)<
        LGT_X(NO_APPL_PS_STMT))
        TARGET = ' *** SUBSTITUENDA-COLLISION.';
   WHEN(LENGTH(SOURCE)-LGT_X(NO_APPL_PS_STMT)-
        LGT_X(NO_APPL_SS_STMT)+LENGTH(Y(NO_APPL_PS_STMT))+
        LENGTH(Y(NO_APPL_SS_STMT))>30)
        TARGET = ' *** TARGET WOULD BE TOO LONG.';
   OTHERWISE
        TARGET = Y(NO_APPL_PS_STMT)||SUBSTR(SOURCE,
                 LGT_X(NO_APPL_PS_STMT)+1,LENGTH(SOURCE)-
                 LGT_X(NO_APPL_PS_STMT)-
                 LGT_X(NO_APPL_SS_STMT))||
                 INVERT(Y(NO_APPL_SS_STMT),0);
  END;
 END;
END;

        RETURN(TARGET);

/*-------------------------------------------------------------*/
/* FUNCTION-PROCEDURE WHICH RETURNS FOR A GIVEN STRING         */
/* (= SOURCE) THE ADDRESS OF AN APPLICABLE DERIVATION-         */
/* STATEMENT WITH A SUBSCRIPT GREATER THAN OR EQUAL TO         */
/* START AND LESS THAN OR EQUAL TO LIMIT. IF NO DERIVA-        */
/* TION-STATEMENT WITHIN THE SPECIFIED RANGE IS APPLICA-       */
/* BLE, 0 IS RETURNED.                                         */
/*    THE ADDRESS OF THE APPLICABLE STATEMENT IS DETER-        */
/* MINED BY A BINARY SEARCH FOR THE LONGEST STRING X_Z(I)      */
/* (START<=I<=LIMIT) WHICH IS A LEFT SUBSTRING OF SOURCE.      */
/*    THE ALGORITHM FOR THE BINARY SEARCH HAS - SLIGHTLY       */
/* MODIFIED - BEEN TAKEN FROM: KNUTH, D. E. (1973): THE        */
/* ART OF COMPUTER PROGRAMMING. VOL. 3. SORTING AND            */
/* SEARCHING. READING (MASS.)/MENLO PARK (CAL.)/LONDON/        */
/* AMSTERDAM/DON MILLS (ONTARIO)/SYDNEY. P. 407-411.           */
/*-------------------------------------------------------------*/

ADAPSTM: PROC(SOURCE,START,LIMIT) RETURNS(BIN FIXED(15));
DCL      SOURCE                CHAR(30) VAR,
         KEY                   CHAR(30) VAR,
         LEFT_OF_KEY           CHAR(30) VAR,
         START                 BIN FIXED(15),
         LIMIT                 BIN FIXED(15),
         L_BDRY                BIN FIXED(15),
         U_BDRY                BIN FIXED(15),
         MIDDLE                BIN FIXED(15),
         STMT_NO               BIN FIXED(15) INIT(0),
         LLCXZ_LST_FND_STM     BIN FIXED(15),
         I                     BIN FIXED(15);
```

```
          IF LIMIT¬=0 THEN
DO;
          L_BDRY             = START;
          KEY                = SOURCE||' ';
          LLCXZ_LST_FND_STM  = LENGTH(KEY);
 DO       I=0 TO LENGTH(KEY) UNTIL(I>=LLCXZ_LST_FND_STM);
          LEFT_OF_KEY        = SUBSTR(KEY,1,I);
          U_BDRY             = LIMIT;
  BIN_SRCH: DO  WHILE(U_BDRY>=L_BDRY);
          MIDDLE             = (L_BDRY+U_BDRY)/2;
   SELECT;
    WHEN(LEFT_OF_KEY>X_Z(MIDDLE)) L_BDRY=MIDDLE+1;
    WHEN(LEFT_OF_KEY<X_Z(MIDDLE)) U_BDRY=MIDDLE-1;
    OTHERWISE SELECT(LENGTH(X_Z(MIDDLE)));
     WHEN(I) DO;
          STMT_NO            = MIDDLE;
          L_BDRY             = MIDDLE+1;
          LLCXZ_LST_FND_STM  = LGT_LGST_CNTNG_XZ(STMT_NO);
          LEAVE BIN_SRCH;
     END;
     WHEN(I-1)       L_BDRY = MIDDLE+1;
     OTHERWISE       U_BDRY = MIDDLE-1;
    END;
   END;
  END;
 END;
END;

          RETURN(STMT_NO);

END       ADAPSIM;

END       DERIVE;
```

```
/*-------------------------------------------------------*/
/* NAME: LISTOUT                                         */
/*                                                       */
/* FUNCTION:                                             */
/* LISTOUT IS A SUBROUTINE-PROCEDURE. IT READS A SOURCE- */
/* TARGET-LIST FROM CORPUS AND WRITES IT WITH A CLEAR    */
/* FORMAT ONTO THE SYSPRINT-FILE.                        */
/*                                                       */
/* FILES:                                                */
/*  INPUT:  CORPUS,                                      */
/*  OUTPUT: SYSPRINT.                                    */
/*-------------------------------------------------------*/

LISTOUT: PROC REORDER;
DCL      1 REC              BASED(REC_PNTR),
          2 DUMMY_1         CHAR(10),
          2 SOURCE          CHAR(30),
          2 TARGET          CHAR(30),
          2 DUMMY_2         CHAR(10),

         PAGENO             BIN FIXED(15) EXT,
         REC_PNTR           PTR,
         CORPUS             FILE RECORD SEQL,
         SYSPRINT           FILE STREAM PRINT;

/*-------------------------------------------------------*/
/* ENDPAGE-CONDITION-ON-UNIT FOR THE OUTPUT OF A HEADING */
/* AT THE START OF A NEW PAGE:                           */
/*-------------------------------------------------------*/

ON       ENDPAGE(SYSPRINT) BEGIN;
         PAGENO = PAGENO+1;
         PUT PAGE EDIT
         ('PAGE ',PAGENO)        (COL(62),A,F(4))
         ('|')                   (SKIP(3),COL(39),A)
         ('SOURCE','|')          (COL(20),A,COL(39),A)
         ('GENERATED TARGET')    (COL(47),A)
         ('|')                   (COL(39),A)
         ((31)'-','+',(31)'-')   (COL(8),3 A)
         ('|')                   (COL(39),A);
END;

         SIGNAL ENDPAGE(SYSPRINT);

/*-------------------------------------------------------*/
/* INPUT AND OUTPUT OF THE SOURCE-TARGET-PAIRS:          */
/*-------------------------------------------------------*/

ON       ENDFILE(CORPUS) GOTO SHUT;
         OPEN FILE(CORPUS) INPUT;
```

```
DO          WHILE('1'B);
            READ FILE(CORPUS) SET(REC_PNTR);
            PUT SKIP EDIT(SOURCE,' | ',TARGET)  (COL(8),3 A);
END;

SHUT:       CLOSE FILE(CORPUS);

END         LISTOUT;
```

```
/*-------------------------------------------------------*/
/* NAME: INVERT                                          */
/*                                                       */
/* FUNCTION:                                             */
/* INVERT IS A FUNCTION-PROCEDURE. IT RETURNS (DEPENDENT */
/* ON THE VALUE OF ITS SECOND ARGUMENT)                  */
/*  - EITHER A STRING WHICH DIFFERS FROM THE STRING GIVEN */
/*    AS ITS FIRST ARGUMENT ONLY IN THAT THE LONGEST LEFT */
/*    SUBSTRING OF NON-BLANK CHARACTERS IS INVERTED       */
/*  - OR THE INVERSION OF THE GIVEN STRING               */
/*  - OR A STRING WHICH DIFFERS FROM THE GIVEN STRING ON- */
/*    LY IN THAT A LEFT SUBSTRING OF A GIVEN LENGTH IS   */
/*    INVERTED.                                          */
/*                                                       */
/* PARAMETERS:                                           */
/*  STRING_1: CHAR(*). STRING TO BE INVERTED. MAY NOT BE */
/*            LONGER THAN 30 CHARACTERS.                 */
/*  SWITCH:   BIN FIXED(15).                             */
/*            - IF SWITCH=-1, THE LONGEST LEFT SUBSTRING */
/*              OF NON-BLANK CHARACTERS IS INVERTED.     */
/*            - IF SWITCH=0, THE WHOLE STRING IS INVER-  */
/*              TED.                                     */
/*            - IF SWITCH>0, THE LEFT SUBSTRING OF LENGTH */
/*              SWITCH IS INVERTED.                      */
/*                                                       */
/* RETURNED VALUE:                                       */
/*  CHAR(30) VAR. DETAILED DESCRIPTION: S. A.            */
/*-------------------------------------------------------*/

INVERT:  PROC(STRING_1,SWITCH)
           RETURNS(CHAR(30) VAR) RECRDER;
DCL      STRING_1                          CHAR(*),
         CUT_STRING_1(LENGTH(STRING_1))    CHAR(1) DEF
                                             STRING_1 POS(1),
         STRING_2                          CHAR(LENGTH(
                                             STRING_1))
                                             INIT(' '),
         CUT_STRING_2(LENGTH(STRING_1))    CHAR(1) DEF
                                             STRING_2 POS(1),
         I                                 BIN FIXED(15),
         LGT                               BIN FIXED(15),
         LGT_P1                            BIN FIXED(15),
         SWITCH                            BIN FIXED(15),
         (INDEX,LENGTH)                    BUILTIN;

SELECT(SWITCH);
 WHEN(-1)  LGT = INDEX(STRING_1||' ',' ')-1;
 WHEN(0)   LGT = LENGTH(STRING_1);
 OTHERWISE LGT = SWITCH;
END;
         LGT_P1 = LGT+1;
DO       I=1 TO LGT;
         CUT_STRING_2(I) = CUT_STRING_1(LGT_P1-I);
END;
```

```
        RETURN(STRING_2);
END INVERT;
```

```
/*-------------------------------------------------------------*/
/* NAME: LLTMTCH                                               */
/*                                                             */
/* FUNCTION:                                                   */
/* LLTMTCH IS A FUNCTION-PROCEDURE. IT RETURNS THE LENGTH      */
/* OF THE LONGEST LEFT MATCH OF 2 GIVEN STRINGS.               */
/*                                                             */
/* PARAMETERS:                                                 */
/*  STRING_1,                                                  */
/*  STRING_2: CHAR(*). 2 STRINGS, WHOSE LENGTH OF THEIR        */
/*            LONGEST LEFT MATCH HAS TO BE DETERMINED.         */
/*            THE STRINGS MAY BE VARYING.                      */
/*  LIMIT:    BIN FIXED(15). SPECIFIES THE POSITION IN         */
/*            STRING_1 AND STRING_2 UP TO WHICH THE            */
/*            LENGTH OF THE LONGEST LEFT MATCH HAS TO BE       */
/*            DETERMINED. IF LIMIT=0, LIMIT IS SET TO          */
/*            MIN(LENGTH(STRING_1),LENGTH(STRING_2)).          */
/*                                                             */
/* RETURNED VALUE:                                             */
/*  BIN FIXED(15). LENGTH OF THE LONGEST LEFT MATCH OF         */
/*  STRING_1 AND STRING_2.                                     */
/*-------------------------------------------------------------*/

LLTMTCH: PROC(STRING_1,STRING_2,LIMIT)
            RETURNS(BIN FIXED(15)) REORDER;
DCL      STRING_1                          CHAR(*),
         CUT_STRING_1(LENGTH(STRING_1))    CHAR(1) DEF STRING_1
                                              POS(1),
         STRING_2                          CHAR(*),
         CUT_STRING_2(LENGTH(STRING_2))    CHAR(1) DEF STRING_2
                                              POS(1),
         LGT                               BIN FIXED(15)
                                              INIT(1),
         LIMIT                             BIN FIXED(15),
         (LENGTH,MIN)                      BUILTIN;

         IF LIMIT=0  THEN LIMIT=MIN(LENGTH(STRING_1),
                                    LENGTH(STRING_2));
DO       LGT=1 TO LIMIT;
         IF CUT_STRING_1(LGT)¬=CUT_STRING_2(LGT) THEN LEAVE;
END;

         RETURN(LGT-1);

END      LLTMTCH;
```

10.2 Ergebnisse der Anwendung von PRISM auf die Motion im Französischen

Die folgenden Seiten illustrieren das Input-Output-Verhalten von PRISM's Lernkomponente am Beispiel der Motion im Französischen. Aufbau, Inhalt und Umfang des vorgegebenen Lernkorpus sind in Abschnitt 6.2 beschrieben. Da die in der französischen Schriftsprache üblichen diakritischen Zeichen nicht auf den verwendeten Ein- und Ausgabegeräten verfügbar waren, mußten sie kodiert werden. Ihre Kodierung ist in folgender Tabelle wiedergegeben:

Diakritisches Zeichen	Kodierung
´	<
`	>
^	/
¨	:

Abb. 10.1

Die Kodierungen stehen immer unmittelbar rechts von dem Buchstaben, auf den sie sich beziehen. 'héroïne' <Heldin> wird also mit 'he<roi:ne' kodiert. Da die Cedille nicht in der Wortpaarliste auftritt, mußte für sie keine Kodierung eingeführt werden.

Wir geben zunächst das eingegebene Lernkorpus wieder, und zwar zuerst alphabetisch *vorläufig* nach Quellstrings sortiert, dann alphabetisch *rückläufig* nach Quellstrings sortiert. Der Abdruck beider Sortierungen erschien uns sinnvoll, da er dem Leser die Ermittlung derjenigen Quellstrings des Lernkorpus erleichtert, auf die die erlernten präfixorientierten und suffixorientierten Anweisungen jeweils anwendbar sind (Zu den präfixorientierten Anweisungen lassen sich die entsprechenden Quellstrings leichter in der vorläufigen Sortierung finden, für die suffixorientierten Anweisungen sollte man die rückläufige Sortierung heranziehen). Wir müssen hier allerdings nachdrücklich davor warnen, aus der Wiedergabe beider Sortierungen die Annahme abzuleiten, daß der Lernkomponente eine vor- *und* eine rückläufige Sortierung des Lernkorpus vorgegeben werden muß: Das Lernkorpus muß und kann grundsätzlich nur einmal vorgegeben werden, und zwar in beliebi-

ger Sortierung.

Im Anschluß an die beiden Sortierungen des Lernkorpus ist die Druckerausgabe von PRISM's Lernkomponente für dieses Lernkorpus wiedergegeben.

Alphabetisch vorläufig nach Quellstrings
sortiertes Lernkorpus

a/ne	a/nesse
abbe<	abbesse
abbonne<	abbonne<e
accuse<	accuse<e
acheteur	acheteuse
acrobate	acrobate
acteur	actrice
actionnaire	actionnaire
administrateur	administratrice
adolescent	adolescente
adversaire	adversaire
agitateur	agitatrice
ai:eul	ai:eule
ambassadeur	ambassadrice
ami	amie
animateur	animatrice
artiste	artiste
assassin	assassine
auditeur	auditrice
aviateur	aviatrice
avocat	avocate
bachelier	bachelie>re
baigneur	baigneuse
bambin	bambine
baron	baronne
basketeur	basketeuse
bateleur	bateleuse
bibliothe<caire	bibliothe<caire
blanchisseur	blanchisseuse
bohe<mien	bohe<mienne
boucher	bouche>re
boulanger	boulange>re
bourgeois	bourgeoise
boutiquier	boutiquie>re
cabaretier	cabaretie>re
cafetier	cafetie>re
caissier	caissie>re
calomniateur	calomniatrice
camarade	camarade
cambrioleur	cambrioleuse
candidat	candidate
cavalier	
chah	
chameau	chamelle
champion	championne
chansonnier	chansonnie>re
chanteur	chanteuse
chapelier	chapelie>re
chasseur	chasseresse
chat	chatte
cheik	
chevalier	

choriste	choriste
chre<tien	chre<tienne
citadin	citadine
citoyen	citoyenne
client	cliente
coiffeur	coiffeuse
collaborateur	collaboratrice
colle<gien	colle<gienne
come<dien	come<dienne
compatriote	compatriote
compositeur	compositrice
concierge	concierge
conciliateur	conciliatrice
conducteur	conductrice
contempteur	contemptrice
conteur	contrice
contro/leur	contro/leuse
coq	
correspondancier	correspondancie>re
corrupteur	corruptrice
coureur	coureuse
cousin	cousine
cre<ancier	cre<ancie>re
cre<ateur	cre<atrice
cuisinier	cuisinie>re
cycliste	cycliste
danseur	danseuse
de<biteur	de<bitrice
de<corateur	de<ccratrice
de<fendeur	de<fenderesse
de<funt	de<funte
de<tenteur	de<tentrice
de<tracteur	de<tractrice
demandeur	demanderesse
dessinateur	dessinatrice
dindon	dinde
directeur	directrice
docteur	doctoresse
dompteur	dompteuse
dresseur	dresseuse
droguiste	droguiste
duc	duchesse
dupeur	dupeuse
e<colier	e<colie>re
e<couteur	e<couteuse
e<cuyer	e<cuye>re
e<diteur	e<ditrice
e<ducateur	e<ducatrice
e<lecteur	e<lectrice
e<meutier	e<meutie>re
e<picier	e<picie>re
e<tranger	e<trange>re

e<tudiant	e<tudiante
employe<	employe<e
empoisonneur	empoisonneuse
ennemi	ennemie
entrepreneur	entrepreneuse
envoye<	envoye<e
escrimeur	escrimeuse
espion	espionne
examinateur	examinatrice
excitateur	excitatrice
expe<diteur	expe<ditrice
expe<rimentateur	expe<rimentatrice
fanatique	fanatique
faneur	faneuse
faubourien	faubourienne
faucheur	faucheuse
fiance<	fiance<e
figurant	figurante
fils	fille
fla/neur	fla/neuse
fonctionnaire	fonctionnaire
fournisseur	fournisseuse
fraudeur	fraudeuse
gagnant	gagnante
garant	garante
gardien	gardienne
ge<ant	ge<ante
ge<rant	ge<rante
gitan	gitane
guerrier	guerrie>re
gueux	gueuse
he<ritier	he<ritie>re
he<ros	he<roi:ne
hivernant	hivernante
ho/te	ho/tesse
horloger	horlcge>re
immigrant	immigrante
incestueux	incestueuse
inculpe<	inculpe<e
indige>ne	indige>ne
infirmier	infirmie>re
instigateur	instigatrice
interlocuteur	interlocutrice
interme<diaire	interme<diaire
jardinier	jardinie>re
joueur	joueuse
journaliste	journaliste
jure<	jure<e
lanceur	lanceuse
lecteur	lectrice
lieutenant	
liftier	liftie>re

lion	lionne
locataire	locataire
logeur	logeuse
loueur	loueuse
loup	louve
lugeur	lugeuse
lutteur	lutteuse
lyce<en	lyce<enne
magicien	magicienne
mahome<tan	mahome<tane
mai/tre	mai/tresse
malfaiteur	malfaitrice
manufacturier	manufacturie>re
marchand	marchande
marquis	marquise
martyr	martyre
me<diateur	me<diatrice
me<tis	me<tisse
mendiant	mendiante
messager	message>re
meunier	meunie>re
meurtrier	meurtrie>re
mise<reux	mise<reuse
modeleur	modeleuse
moissoneur	moissoneuse
moniteur	monitrice
montagnard	montagnarde
monteur	monteuse
mort	morte
motocycliste	motocycliste
mula/tre	mula/tresse
musulman	musulmane
nageur	nageuse
nain	naine
narrateur	narratrice
ne<cessiteux	ne<cessiteuse
ne<gociateur	ne<gociatrice
ne>gre	ne<gresse
observateur	observatrice
oncle	
ope<rateur	ope<ratrice
orateur	oratrice
orphelin	orpheline
ours	ourse
ouvreur	ouvreuse
ouvrier	ouvrie>re
parrain	marraine
partisan	partisane
patineur	patineuse
patriote	patriote
patron	patronne
paysan	paysanne

pe<cheur	pe<cheresse
pe<daleur	pe<daleuse
pe/cheur	pe/cheuse
pe>lerin	pe>lerine
pe>re	me>re
peintre	peintresse
pelletier	pelletie>re
perdant	perdante
pianiste	pianiste
pie<ton	pie<tonne
pilleur	pilleuse
plongeur	plongeuse
poe>te	poe<tesse
poupon	pouponne
pre<sentateur	pre<sentatrice
pre<sident	pre<sidente
prince	princesse
prisonnier	prisonnie>re
profanateur	profanatrice
promeneur	promeneuse
promoteur	promotrice
proprie<taire	proprie<taire
provocateur	provocatrice
rameur	rameuse
rapatrie<	rapatrie<e
re<clusionnaire	re<clusionnaire
re<volutionnaire	re<volutionnaire
rebelle	rebelle
receleur	receleuse
ressortissant	ressortissante
retraite<	retraite<e
riverain	riveraine
ro/deur	ro/deuse
roi	reine
romancier	romancie>re
saint	sainte
sauteur	sauteuse
savant	savante
se<ditieux	se<ditieuse
se<ducteur	se<ductrice
se<journant	se<journante
secre<taire	secre<taire
serf	serve
serveur	serveuse
serviteur	servante
skieur	skieuse
soldat	soldate
sorcier	sorcie>re
souffleur	souffleuse
souverain	souveraine
spe<culateur	spe<culatrice
speaker	speakerine

spectateur	spectatrice
sportif	sportive
ste<nographe	ste<nographe
steward	stewardess
stoppeur	stoppeuse
suborneur	suborneuse
sultan	sultane
supple<ant	supple<ante
tailleur	
tentateur	tentatrice
the<oricien	the<oricienne
tigre	tigresse
tisseur	tisseuse
touriste	touriste
traducteur	traductrice
trage<dien	trage<dienne
trai/tre	trai/tresse
travailleur	travailleuse
tricheur	tricheuse
trompeur	trompeuse
tsar	tsarine
tsigane	tsigane
usager	usage>re
usurier	usurie>re
utilisateur	utilisatrice
vacher	vache>re
vagabond	vagabonde
ve<rificateur	ve<rificatrice
vendangeur	vendangeuse
vendeur	vendeuse
vengeur	vengeresse
veuf	veuve
vigneron	vigneronne
villageois	villageoise
violateur	viclatrice
violoniste	violoniste
virtuose	virtuose
voisin	voisine
voleur	voleuse
voyageur	voyageuse

Alphabetisch rückläufig nach Quellstrings sortiertes Lernkorpus

abbe<	abbesse
fiance<	fiance<e
rapatrie<	rapatrie<e
abbonne<	abbonne<e
inculpe<	inculpe<e
jure<	jure<e
accuse<	accuse<e
retraite<	retraite<e
employe<	employe<e
envoye<	envoye<e
duc	duchesse
marchand	marchande
vagabond	vagabonde
montagnard	montagnarde
steward	stewardess
prince	princesse
camarade	camarade
concierge	concierge
ste<nographe	ste<nographe
oncle	
rebelle	rebelle
a/ne	a/nesse
indige>ne	indige>ne
tsigane	tsigane
pe>re	me>re
ne>gre	ne<gresse
tigre	tigresse
bibliothe<caire	bibliothe<caire
interme<diaire	interme<diaire
re<clusionnaire	re<clusionnaire
actionnaire	actionnaire
fonctionnaire	fonctionnaire
re<volutionnaire	re<volutionnaire
adversaire	adversaire
proprie<taire	proprie<taire
secre<taire	secre<taire
locataire	locataire
mula/tre	mula/tresse
mai/tre	mai/tresse
trai/tre	trai/tresse
peintre	peintresse
virtuose	virtuose
ho/te	ho/tesse
poe>te	poe<tesse
acrobate	acrobate
patriote	patriote
compatriote	compatriote
journaliste	journaliste
cycliste	cycliste
motocycliste	motocycliste
pianiste	pianiste
violoniste	violoniste

choriste	choriste
touriste	touriste
artiste	artiste
droguiste	droguiste
fanatique	fanatique
sportif	sportive
serf	serve
veuf	veuve
chah	
ami	amie
ennemi	ennemie
roi	reine
cheik	
ai:eul	ai:eule
musulman	musulmane
partisan	partisane
paysan	paysanne
mahome<tan	mahome<tane
gitan	gitane
sultan	sultane
lyce<en	lyce<enne
magicien	magicienne
the<oricien	the<oricienne
trage<dien	trage<dienne
come<dien	come<dienne
gardien	gardienne
colle<gien	colle<gienne
bohe<mien	bohe<mienne
faubourien	faubourienne
chre<tien	chre<tienne
citoyen	citoyenne
nain	naine
riverain	riveraine
souverain	souveraine
parrain	marraine
bambin	bambine
citadin	citadine
orphelin	orpheline
pe>lerin	pe>lerine
voisin	voisine
assassin	assassine
cousin	cousine
dindon	dinde
lion	lionne
champion	championne
espion	espionne
poupon	pouponne
baron	baronne
vigneron	vigneronne
patron	patronne
pie<ton	pie<tonne
loup	louve

coq	
tsar	tsarine
messager	message>re
usager	usage>re
boulanger	boulange>re
e<tranger	e<trange>re
horloger	horloge>re
vacher	vache>re
boucher	bouche>re
e<picier	e<picie>re
cre<ancier	cre<ancie>re
correspondancier	correspondancie>re
romancier	romancie>re
sorcier	sorcie>re
cavalier	
chevalier	
bachelier	bachelie>re
chapelier	chapelie>re
e<colier	e<colie>re
infirmier	infirmie>re
jardinier	jardinie>re
cuisinier	cuisinie>re
prisonnier	prisonnie>re
chansonnier	chansonnie>re
meunier	meunie>re
guerrier	guerrie>re
meurtrier	meurtrie>re
usurier	usurie>re
manufacturier	manufacturie>re
ouvrier	ouvrie>re
caissier	caissie>re
cafetier	cafetie>re
pelletier	pelletie>re
cabaretier	cabaretie>re
liftier	liftie>re
he<ritier	he<ritie>re
e<meutier	e<meutie>re
boutiquier	boutiquie>re
speaker	speakerine
e<cuyer	e<cuye>re
lanceur	lanceuse
ro/deur	ro/deuse
ambassadeur	ambassadrice
demandeur	demanderesse
de<fendeur	de<fenderesse
vendeur	vendeuse
fraudeur	fraudeuse
coiffeur	coiffeuse
nageur	nageuse
voyageur	voyageuse
vendangeur	vendangeuse
vengeur	vengeresse

plongeur	plongeuse
logeur	logeuse
lugeur	lugeuse
pe<cheur	pe<cheresse
pe/cheur	pe/cheuse
tricheur	tricheuse
faucheur	faucheuse
skieur	skieuse
contro/leur	contro/leuse
pe<daleur	pe<daleuse
receleur	receleuse
modeleur	modeleuse
bateleur	bateleuse
souffleur	souffleuse
tailleur	
travailleur	travailleuse
pilleur	pilleuse
cambrioleur	cambrioleuse
voleur	voleuse
rameur	rameuse
escrimeur	escrimeuse
fla/neur	fla/neuse
faneur	faneuse
promeneur	promeneuse
entrepreneur	entrepreneuse
baigneur	baigneuse
patineur	patineuse
empoisonneur	empoisonneuse
moissoneur	moissoneuse
suborneur	suborneuse
trompeur	trompeuse
stoppeur	stoppeuse
dupeur	dupeuse
coureur	coureuse
ouvreur	ouvreuse
danseur	danseuse
chasseur	chasseresse
dresseur	dresseuse
blanchisseur	blanchisseuse
fournisseur	fournisseuse
tisseur	tisseuse
cre<ateur	cre<atrice
ve<rificateur	ve<rificatrice
provocateur	provocatrice
e<ducateur	e<ducatrice
instigateur	instigatrice
ne<gociateur	ne<gociatrice
me<diateur	me<diatrice
conciliateur	conciliatrice
calomniateur	calomniatrice
aviateur	aviatrice
violateur	violatrice

spe<culateur	spe<culatrice
animateur	animatrice
profanateur	profanatrice
examinateur	examinatrice
dessinateur	dessinatrice
ope<rateur	ope<ratrice
orateur	oratrice
collaborateur	collaboratrice
de<corateur	de<coratrice
narrateur	narratrice
administrateur	administratrice
utilisateur	utilisatrice
spectateur	spectatrice
excitateur	excitatrice
agitateur	agitatrice
expe<rimentateur	expe<rimentatrice
pre<sentateur	pre<sentatrice
tentateur	tentatrice
observateur	observatrice
acteur	actrice
de<tracteur	de<tractrice
lecteur	lectrice
e<lecteur	e<lectrice
directeur	directrice
docteur	doctoresse
se<ducteur	se<ductrice
traducteur	traductrice
conducteur	conductrice
acheteur	acheteuse
basketeur	basketeuse
malfaiteur	malfaitrice
de<biteur	de<bitrice
e<diteur	e<ditrice
expe<diteur	expe<ditrice
auditeur	auditrice
moniteur	monitrice
compositeur	compositrice
serviteur	servante
chanteur	chanteuse
de<tenteur	de<tentrice
conteur	contrice
monteur	monteuse
promoteur	promotrice
contempteur	contemptrice
dompteur	dompteuse
corrupteur	corruptrice
lutteur	lutteuse
sauteur	sauteuse
interlocuteur	interlocutrice
e<couteur	e<couteuse
joueur	joueuse
loueur	loueuse

serveur	serveuse
martyr	martyre
villageois	villageoise
bourgeois	bourgeoise
me<tis	me<tisse
marquis	marquise
fils	fille
he<ros	he<roi:ne
ours	ourse
avocat	avocate
candidat	candidate
soldat	soldate
chat	chatte
ge<ant	ge<ante
supple<ant	supple<ante
perdant	perdante
mendiant	mendiante
e<tudiant	e<tudiante
lieutenant	
gagnant	gagnante
hivernant	hivernante
se<journant	se<journante
ge<rant	ge<rante
garant	garante
immigrant	immigrante
figurant	figurante
ressortissant	ressortissante
savant	savante
adolescent	adolescente
pre<sident	pre<sidente
client	cliente
saint	sainte
de<funt	de<funte
mort	morte
chameau	chamelle
se<ditieux	se<ditieuse
mise<reux	mise<reuse
ne<cessiteux	ne<cessiteuse
gueux	gueuse
incestueux	incestueuse

Druckerausgabe der Lernkomponente

DATE: 19.12.83 TIME: 14:06:28 PAGE 1

```
PPPPPPPP   RPRRRRRRR   IIIIIIIIII   SSSSSSSS   M         M
PPPPPPPPP  RRRRRRRRRR  IIIIIIIIII  SSSSSSSSSS  MM       MM
PP     PP  RR      RR      II      SS      SS  MMMM     MMMM
PP     PP  RR      RR      II      SS          MMMMM   MMMMM
PP     PP  RR      RR      II      SSS         MM MMMMM MM
PPPPPPPPP  RRRRRRRRRR      II       SSSSSSS    MM  MM  MM
PPPPPPPP   RRRRRRRRR       II         SSSSSSS  MM      MM
PP         RR  RR          II              SSS MM      MM
PP         RR   RR         II              SS  MM      MM
PP         RR    RR        II      SS      SS  MM      MM
PP         RR     RR   IIIIIIIIII  SSSSSSSSSS MM      MM
PP         RR      RR  IIIIIIIIII   SSSSSSSS  MM      MM
```

```
 V   V  EEEEE  RRRR   SSSSS  IIIII  OOOOO  N   N      11
 V   V  E      R   R  S        I    O   O  NN  N     1 1
 V   V  EEE    RRRR   SSSSS    I    O   O  N N N     1  1
  V V   E      R R        S    I    O   O  N  NN       1
   V    EEEEE  R  R   SSSSS  IIIII  OOOOO  N   N       1
```

AUTHOR: K. WOTHKE
INSTITUT FUER KOMMUNIKATIONS-
FORSCHUNG UND PHONETIK
UNIVERSITY OF BONN
POPPELSDORFER ALLEE 47

53 BONN 1

GERMANY (WEST)

GIVEN OPTION-FIELD:

'LEARN

ERROR-MESSAGES FOR THE OPTION-FIELD:

THE OPTION-FIELD DOES NOT CONTAIN ANY FORMAL ERRORS.

OPTIONS SPECIFIED:

LEARN

OPTIONS USED:

LEARN
PRINT
STATISTICS

ERROR-MESSAGES FOR THE LEARNING-CORPUS:

THE LEARNING-CORPUS DOES NOT CONTAIN ANY FORMAL ERRORS.

GENERATED DERIVATION-STATEMENTS:

SOURCES IN THE LEARNING-CORPUS TO WHICH THE STATEMENT IS APPLICABLE		STATEMENT
P. C.	ABS.	
0.67%	2	#'che'¬
0.33%	1	#'lie'¬
0.33%	1	#'on'¬
0.33%	1	#'ta'¬
0.33%	1	'h'#¬
0.33%	1	'q'#¬
0.33%	1	'alier'#¬
0.33%	1	'ne>' -> 'ne<'/#__
0.67%	2	'p' -> 'm'/#__('arr'|'e>r')
96.33%	289	'' -> ''/#__
0.33%	1	'au' -> 'lle'/__#
0.33%	1	's' -> 'i:ne'/'o'__#
0.33%	1	's' -> 'le'/'l'__#
0.33%	1	'iteur' -> 'ante'/'v'__#
0.33%	1	'eur' -> 'oresse'/'oct'__#
0.33%	1	'p' -> 've'/__#
0.33%	1	'on' -> 'e'/'d'__#
0.33%	1	'oi' -> 'eine'/__#
0.33%	1	'>te' -> '<tesse'/__#
0.33%	1	'<' -> 'sse'/'be'__#
0.33%	1	'' -> 'ess'/'ward'__#
0.33%	1	'' -> 'te'/'hat'__#
0.33%	1	'' -> 'hesse'/'c'__#
0.33%	1	'' -> 'se'/'tis'__#
0.67%	2	'' -> 'ine'/('ar'|'ker')__#
1.00%	3	'f' -> 've'/__#
1.67%	5	'x' -> 'se'/__#
1.67%	5	'ur' -> 'resse'/('ande'|'fende'|'enge'|'<che'|'asse')__#
3.00%	9	'' -> 'sse'/('ce'|'/ne'|'gre'|'tre'|'/te')__#
6.67%	20	'' -> 'ne'/('ysan'|'en'|'on')__#
10.33%	31	'' -> ''/'e'__#
11.67%	35	'r' -> '>re'/'e'__#
17.33%	52	'eur' -> 'rice'/('ad'|'at'|'ct'|'it'|'ent'|'cont'|'ot'|'empt'|'upt'|'cut')__#
19.00%	57	'r' -> 'se'/'u'__#
19.67%	59	'' -> 'e'/__#

NUMBER OF SOURCE-TARGET-PAIRS
IN THE LEARNING-CORPUS: 300
NUMBER OF GENERATED STATEMENTS: 35

INFORMATORY MESSAGE:

THE GENERATED DERIVATION-STATEMENTS WERE STORED IN THE DATA SET ALLOCATED TO FILE STMTFLE.

```
EEEEE  N   N  DDDD          OOOOO  FFFFF
E      NN  N  D   D         O   O  F
EEE    N N N  D   D         O   O  FFF
E      N  NN  D   D         O   O  F
EEEEE  N   N  DDDD          OOOOO  F
```

```
PPPPPPPP   RRRRRRRRR   IIIIIIIIII   SSSSSSSS   M         M
PPPPPPPPP  RRRRRRRRRR  IIIIIIIIII  SSSSSSSSSS  MM       MM
PP     PP  RR      RR      II      SS      SS  MMMM     MMMM
PP     PP  RR      RR      II      SS          MMMMM   MMMMM
PP     PP  RR      RR      II      SSS         MM MMMM MM
PPPPPPPPP  RRRRRRRRRR      II       SSSSSSS    MM  MM  MM
PPPPPPPP   RRRRRRRRR       II         SSSSSSS  MM      MM
PP         RR  RR          II              SSS MM      MM
PP         RR   RR         II               SS MM      MM
PP         RR    RR        II      SS       SS MM      MM
PP         RR     RR   IIIIIIIIII  SSSSSSSSSS MM      MM
PP         RR      RR  IIIIIIIIII   SSSSSSSS  MM      MM
```

10.3 Ergebnisse der Anwendung von PRISM auf die Negation englischer Adjektive

Als zweites Anwendungsbeispiel von PRISM's Lernkomponente sei in diesem Abschnitt die Erlernung der Regeln zur Negation englischer Adjektive mit dem Präfix 'in' und seinen Varianten 'im', 'il' und 'ir' wiedergegeben. Es folgen nacheinander das eingegebene Lernkorpus und die Druckerausgabe der Lernkomponente.

Lernkorpus

accessible	inaccessible
cautious	incautious
commutable	incommutable
legible	illegible
mature	immature
moral	immoral
perfect	imperfect
possible	impossible
regular	irregular
tolerable	intolerable
variable	invariable

Druckerausgabe der Lernkomponente

DATE: 05.06.84 TIME: 19:14:43 PAGE 1

```
PPPPPPPP   RRRRRRRRR   IIIIIIIIII   SSSSSSSS   M         M
PPPPPPPPP  RRRRRRRRRR  IIIIIIIIII  SSSSSSSSSS  MM       MM
PP      PP RR       RR     II      SS      SS  MMMM     MMMM
PP      PP RR       RR     II      SS          MMMMM   MMMMM
PP      PP RR       RR     II      SSS         MM MMMMM MM MM
PPPPPPPPP  RRRRRRRRRR     II       SSSSSSS    MM  MM   MM MM
PPPPPPPP   RRRRRRRRR      II         SSSSSSS  MM       MM  MM
PP         RR  RR         II               SSS MM       MM  MM
PP         RR   RR        II               SS  MM       MM  MM
PP         RR    RR       II      SS       SS  MM       MM  MM
PP         RR     RR  IIIIIIIIII  SSSSSSSSSS  MM       MM  MM
PP         RR      RR IIIIIIIIII   SSSSSSSS   MM       MM  MM


 V   V  EEEEE  RRRR   SSSSS  IIIII  OOOOO  N   N      11
 V   V  E      R   R  S        I    O   O  NN  N    1 1
 V   V  EEE    RRRR   SSSSS    I    O   O  N N N    1  1
  V V   E      R R        S    I    O   O  N  NN      1
   V    EEEEE  R  R   SSSSS  IIIII  OOOOO  N   N      1
```

AUTHOR: K. WOTHKE
INSTITUT FUER KOMMUNIKATIONS-
FORSCHUNG UND PHONETIK
UNIVERSITY OF BONN
POPPELSDORFER ALLEE 47

53 BONN 1

GERMANY (WEST)

GIVEN OPTION-FIELD:

'LEARN

ERROR-MESSAGES FOR THE OPTION-FIELD:

THE OPTION-FIELD DOES NOT CONTAIN ANY FORMAL ERRORS.

OPTIONS SPECIFIED:

LEARN

OPTIONS USED:

LEARN
PRINT
STATISTICS

ERROR-MESSAGES FOR THE LEARNING-CORPUS:

THE LEARNING-CORPUS DOES NOT CONTAIN ANY FORMAL ERRORS.

GENERATED DERIVATION-STATEMENTS:

SOURCES IN THE LEARNING-CORPUS TO WHICH THE STATEMENT IS APPLICABLE		STATEMENT
P. C.	ABS.	
9.09%	1	'' -> 'il'/#__'l'
9.09%	1	'' -> 'ir'/#__'r'
36.36%	4	'' -> 'im'/#__('m'|'p')
45.45%	5	'' -> 'in'/#__
100.00%	11	'' -> ''/__#

NUMBER OF SOURCE-TARGET-PAIRS
IN THE LEARNING-CORPUS: 11
NUMBER OF GENERATED STATEMENTS: 5

INFORMATORY MESSAGE:

THE GENERATED DERIVATION-STATEMENTS WERE STORED IN THE DATA SET ALLOCATED TO FILE STMTFLE.

```
EEEEE  N   N  DDDD          OOOOO  FFFFF
E      NN  N  D   D         O   O  F
EEE    N N N  D   D         O   O  FFF
E      N  NN  D   D         O   O  F
EEEEE  N   N  DDDD          OOOOO  F
```

```
PPPPPPPP   RRRRRRRRR   IIIIIIIIII   SSSSSSSS   M         M
PPPPPPPPP  RRRRRRRRRR  IIIIIIIIII  SSSSSSSSSS  MM       MM
PP      PP RR       RR     II      SS       SS MMMM     MMMM
PP      PP RR       RR     II      SS          MMMMM   MMMMM
PP      PP RR       RR     II      SSS         MM MMMMM MM
PPPPPPPPP  RRRRRRRRRR      II       SSSSSSS    MM  MM   MM
PPPPPPPP   RRRRRRRRR       II         SSSSSSS  MM       MM
PP         RR  RR          II              SSS MM       MM
PP         RR   RR         II               SS MM       MM
PP         RR    RR        II      SS       SS MM       MM
PP         RR     RR  IIIIIIIIII  SSSSSSSSSS  MM       MM
PP         RR      RR IIIIIIIIII   SSSSSSSS   MM       MM
```

LITERATURVERZEICHNIS

Althaus, H. P. (1973a): Graphemik. In: Althaus/Henne/Wiegand (1973: 118-132).
Althaus, H. P. (1973b): Graphetik. In: Althaus/Henne/Wiegand (1973: 105-110).
Althaus, H. P./Henne, H./Wiegand, H. E. (Hrsg.) (1973): Lexikon der germanistischen Linguistik. Tübingen.
Bátori, I./Krause, J./Lutz, H. D. (Hrsg.) (1982): Linguistische Datenverarbeitung. Versuch einer Standortbestimmung im Umfeld von Informationslinguistik und Künstlicher Intelligenz. Tübingen.
Bauknecht, K./Zehnder, C. A. (1980): Grundzüge der Datenverarbeitung. Methoden und Konzepte für die Anwendungen. Stuttgart.
Beeh, V. (1981): Sprache und Spracherlernung. Unter mathematisch-biologischer Perspektive. Berlin/New York.
Bergenholtz, H./Mugdan, J. (1979): Einführung in die Morphologie. Stuttgart/Berlin/Köln/Mainz.
Berko, J. (1958): The child's learning of English morphology. In: Word 2/3. S. 150-177.
Biermann, A. W./Feldman, J. A. (1972): A survey of results in grammatical inference. In: Watanabe (1972: 31-54).
Braun, S. (1974): Algorithmische Linguistik. Stuttgart/Berlin/Köln/Mainz.
Bungarten, Th. (1978): Die Überführung ausgewählter Verben und ihrer Valenzen zu Deverbativgruppen in der Nominaltransformation (konzipiert for einen linguistischen Transformationsalgorithmus in einem deutschsprachigen Informations- und Problemlösungssystem (ISLIB)). In: Grazer linguistische Studien 8. S. 5-30.
Bünting, K.-D. (1966): Zur Erzeugung deutscher Wörter mit einem Computer. In: IPK-Forschungsberichte 5. Bonn.
Bünting, K.-D. (1969): Morphologische Strukturen deutscher Wörter: Ein Problem der linguistischen Datenverarbeitung. Bonn. (Diss.)
Chomsky, N. (1965): Aspects of the theory of syntax. Cambridge (Mass.).
Cohen, P. R./Feigenbaum, E. A. (Hrsg.) (1982): The handbook of artificial intelligence. Bd. 3. London.
Derwing, B. L. (1976): Morpheme recognition and the learning of rules for derivational morphology. In: The Canadian journal of linguistics 1. S. 38-66.
Derwing, B. L./Baker, W. J. (1977): The psychological basis for morphological rules. In: Macnamara (1977: 85-110).
Derwing, B. L./Baker, W. J. (1979): Recent research on the acquisition of English morphology. In: Fletcher/Garman (1979: 209-223).

Duden (1973): Grammatik der deutschen Gegenwartssprache. Bearbeitet v. P. Grebe u. a. 3., neu bearbeitete und erweiterte Aufl. Mannheim/Wien/Zürich.
Duden (1976ff.): Das große Wörterbuch der deutschen Sprache in sechs Bänden. 6 Bde. Mannheim/Wien/Zürich.
Fischer, W. (1982): Französischer Wortschatz in Sachgruppen. 6. Aufl. München.
Fleischer, W. (1974): Wortbildung der deutschen Gegenwartssprache. 3., überarb. Aufl. Leipzig.
Fletcher, P./Garman, M. (Hrsg.) (1979): Language acquisition. Studies in first language development. Cambridge/London/New York/New Rochelle/Melbourne/Sydney.
Funk-Kolleg Sprache (1975): Eine Einführung in die moderne Linguistik. Bd. 1. 4. Aufl. Frankfurt (Main).
Gillan, D. J. (1981): Ascent of apes. In: Human and animal mind. 25th Dahlem Workshop. o. O.
Gold, E. M. (1967): Language identification in the limit. In: Information and control 5. S. 447-474.
Graitson, M. (1975): Identification et transformation automatique des morphèmes terminaux dans le lexique médical français. In: Cahiers de lexicologie. Revue internationale de lexicologie et de lexicographie 1. S. 85-109.
Grevisse, M. (1964): Le bon usage. 8. Aufl. Gembloux/Paris.
Gross, M. (1972): Mathematical models in linguistics. Englewood Cliffs (N. J.).
Henzen, W. (1947): Deutsche Wortbildung. Halle (Saale).
Heß, K./Brustkern, J./Lenders, W. (1983): Maschinenlesbare deutsche Wörterbücher. Dokumentation, Vergleich, Integration. Tübingen.
Hoeppner, W. (1980): Derivative Wortbildung der deutschen Gegenwartssprache und ihre algorithmische Analyse. Tübingen.
IBM (1975): OS/VS Utilities. IBM Form GC35-0005-4. 5. Aufl. San Jose (Cal.).
IBM (1976a): OS PL/1 checkout and optimizing compilers: Language reference manual. IBM Form GC33-0009-4. 5. Aufl. San Jose (Cal.).
IBM (1976b): OS PL/1 optimizing compiler: programmer's guide. IBM Form SC33-0006-4. 4. Aufl. San Jose (Cal.).
IBM (1976c): OS/VS2 TSO. Command language reference. IBM Form GC28-0646-3. 4., überarbeitete Aufl. Poughkeepsie (N. Y.)
IBM (1978): OS PL/1 optimizing compiler: messages. IBM Form SC33-0027-4. 5. Aufl. San Jose (Cal.).
IBM (1979): OS/VS2 MVS JCL. IBM Form GC28-0692-4. Poughkeepsie (N. Y.).
Juilland, A. (1965): Dictionnaire inverse de la langue française. London/Den Haag/Paris.
Kamp, H./Pudlatz, H. (1974): Einführung in die Programmiersprache PL/1. Skriptum für Hörer aller Fachrichtungen ab 1. Semester. 2., verb. Aufl. Braunschweig.
Kay, M. (1974): Morphological analysis. o. O. (Vervielfältigtes Vortragsmanuskript auf der 3. Summerschool über computational and mathematical linguistics in Pisa, Juli 1974).
Klaus, G./Buhr, M. (Hrsg.) (1974): Philosophisches Wörterbuch. 2 Bde. 10., neubearbeitete u. erw. Aufl. Leipzig.
Klein, H. W./Strohmeyer, F. (1979): Französische Sprachlehre. Stuttgart.

Klix, F. (1982): Are learning processes evolutionary invariant? An unproved assumption in psychology of learning revisited. In: Zeitschrift für Psychologie 4. S. 381-391.
Knuth, D. E. (1968): The art of computer programming. Bd. 1. Fundamental algorithms. Reading (Mass.)/Menlo Park (Cal.)/ London/Don Mills (Ontario).
Knuth, D. E. (1973): The art of computer programming. Bd. 3. Sorting and searching. Reading (Mass.)/Menlo Park (Cal.)/ London/Amsterdam/Don Mills (Ontario)/Sydney.
Krallmann, D. (Hrsg.) (1978): Kolloquium zur Lage der Linguistischen Datenverarbeitung. Essen. (Interne Vervielfältigung des Vereins LDV-Fittings e. V.)
Krause, J. (1982): 'Prozessuale Linguistik' und Linguistische Informationswissenschaft. Theoretisch-Methodisches zum Verhältnis von linguistischer Sprachanalyse und maschineller Sprachverarbeitung. In: Bátori/Krause/Lutz (1982: 57-69).
Krause, W./Willée, G. (1981): Lemmatizing German newspaper texts with the aid of an algorithm. In: Computers and the humanities 2. S. 101-113.
Kreyszig, E. (1979): Statistische Methoden und ihre Anwendungen. 7. Aufl. Göttingen.
Krumnack, U. (1972): Kriterien zur Klassifizierung von Segmenten für die automatische Analyse und Synthese. In: IPK-Forschungsberichte. Bd. 43. Hamburg. S. 67-148.
Kuhlen, R. (1977): Experimentelle Morphologie in der Informationswissenschaft. München.
Kürschner, W. (1975a): Generative Wortbildung I. Oberflächentypen und Erzeugungsprozess der Komposition. In: Funk-Kolleg Sprache (1975: 337-350).
Kürschner, W. (1975b): Generative Wortbildung II. Funktionstypen und Erzeugungsprozess der Derivation. In: Funk-Kolleg Sprache (1975: 351-364).
Lange-Kowal, E. E./Beaucaire, L. (1977): Langenscheidts Handwörterbuch Französisch. 3. Aufl. Berlin/München.
Lewandowski, Th. (1975): Linguistisches Wörterbuch. Bd. 2 u. 3. Heidelberg.
Lewandowski, Th. (1976): Linguistisches Wörterbuch. Bd. 1. 2., durchgesehene u. erweiterte Aufl. Heidelberg.
Lindsay, P. H./Norman, D. A. (1977): Human information processing: an introduction to psychology. 2. Aufl. New York/San Francisco/ London.
Lyons, J. (1968): Introduction to theoretical linguistics. Cambridge.
Mackensen, L. (1955): Deutsches Wörterbuch. 3. Aufl. Baden-Baden.
Mackensen, L. (1962): Deutsches Wörterbuch. 4. verb. u. erw. Aufl. Baden-Baden.
Macnamara, J. (Hrsg.) (1977): Language learning and thought. London/New York/Toronto/Sydney/San Francisco.
Marchand, H. (1969): The categories and types of present-day English word-formation. A synchronic-diachronic approach. 2., vollst. überarb. u. erw. Aufl. München.
Mater, E. (1966): Deutsche Verben. Bd. 1. Alphabetisches Gesamtverzeichnis. Leipzig.
McCawley, J. D. (1968): Can you count pluses and minuses before you can count? In: Chicago journal of linguistics 2. S. 51-56.

Minsky, M. L. (1968a): Introduction. In: Minsky, M. L. (1968b: 1-32).
Minsky, M. L. (Hrsg.) (1968b): Semantic information processing. Cambridge (Mass.)/London.
Müller, B. S. (1981): Regelgesteuerte Umsetzung von deutschen Texten in gesprochene Sprache für das Sprachausgabegerät VOTRAX. Bonn. (Selbstverlag der Gesellschaft für Mathematik und Datenverarbeitung)
Nida, E. A. (1978): Morphology. The descriptive analysis of words. 2. Aufl. Ann Arbor.
Oakey, S./Cawthorn, R. C. (1981): Inductive learning of pronunciation rules by hypothesis testing and correction. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence. August 1981. Bd. 1. S. 109-114.
Pacak, M. G./Pratt, A. W. (1978): Identification and transformation of terminal morphemes in medical English. Part 2. In: Methods of information in medicine 2. S. 95-100.
Pinker, S. (1979): Formal models of language learning. In: Cognition 3. S. 217-283.
Polenz, P. v. (1973): Synpleremik I: Wortbildung. In: Althaus/Henne/Wiegand (1973: 145-163).
Pratt, A. W./Pacak, M. G. (1969): Identification and transformation of terminal morphemes in medical English. In: Methodik der Information in der Medizin 2. S. 84-90.
The principles of the international phonetic association. London 1949. (Reprint 1977)
Pulgram, E. (1951): Phoneme and grapheme: a parallel. In: Word 1. S. 15-20.
Reeker, L. H. (1976): The computational study of language acquisition. In: Rubinoff/Yovits (1976: 181-237).
Ring, H. (1978): PELIKAN - ein Lernsystem für linguistische Klassifikationsalgorithmen. In: Nachrichten für Dokumentation 6. S. 224-226.
Rubinoff, M./Yovits, M. C. (Hrsg.) (1976): Advances in computers. Bd. 15. New York.
Schippan, Th. (1967): Die Verbalsubstantive der deutschen Sprache der Gegenwart. Leipzig. (Habil.-Schr.)
Seelbach, D. (1975): Computerlinguistik und Dokumentation. München.
Sussman, G. J. (1973): A computational model of skill acquisition. Cambridge (Mass.).
Tesnière, L. (1953): Esquisse d'une syntaxe structurale. Paris.
Uhr, L. (1973): Pattern recognition, learning, and thought. Computer-programmed models of higher mental processes. Englewood Cliffs (N. J.).
Ungeheuer, G. (1977): Konzeption eines Informationssystems auf linguistischer Basis (ISLIB). In: Sprache und Datenverarbeitung 1. S. 46-53.
Viets, D. (1981): Derivationsrelationen und INFO. Zwei Systemkomponenten zur Verbesserung der Retrievalergebnisse im Projekt JUDO-DS. Vorträge zur Beiratssitzung am 9./10./11. September 1981 in Saarbrücken. o. O. (Interne Vervielfältigung)
Wahlster, W. (1982): Aufgaben, Standards und Perspektiven sprachorientierter KI-Forschung. In: Bátori/Krause/Lutz (1982: 13-24).

Wahrig, G. u. a. (Hrsg.) (1977): Deutsches Wörterbuch. Überarb. Aufl. Gütersloh/Berlin.
Watanabe, S. (Hrsg.) (1972): Frontiers of pattern recognition. New York/London.
Weinberg, G. M. (1970): PL/1 programming. A manual of style. New York/St. Louis/San Francisco/Düsseldorf/London/Mexico/ Panama/Sydney/Toronto.
Whitlow Computer Systems (1980): A programmer's guide to SyncSort OS & OS/VS. Englewood Cliffs (N. J.).
Wieland, U./Haller, H. (1978): Verarbeitung natürlichsprachlicher Informationen im Projekt CONDOR (COmmunication in Natürlicher Sprache mit DialogOrientiertem Retrievalsystem). In: Krallmann (1978: 230-259).
Wirth, N. (1983): Algorithmen und Datenstrukturen. 3., überarb. Aufl. Stuttgart.
Wolf, E. (1977): Vom Buchstaben zum Laut. Maschinelle Erzeugung und Erprobung von Umsetzautomaten am Beispiel Schriftenglisch - Phonologisches Englisch. Braunschweig.
Wothke, K. (1984): PRISM User's Guide. Bonn. (= IKP-Arbeitsbericht Nr. 5)
Zimmermann, H. (1978): Automatische Textanalyse und Indexierung (LDV-Aktivitäten in Regensburg - 1. Teil). In: Krallmann (1978: 20-33).

SACHREGISTER

Unterstrichene Seitenzahlen verweisen auf solche Stellen, an denen zu dem jeweiligen Stichwort detailliertere Ausführungen (z. B. auch eine Definition) zu finden sind.

Ableitung, implizite 63, 69
Ableitungsbeziehung, morphologische 62f., 64-69
Ableitungsregel, morphologische 61
Adäquatheit 19, 43, 45f., 48f., 54f., 85, 170
-, schwache 85, 108, 110, 144ff.
-, starke 85, 164ff.
Analogieschluß 2, 3, 9f., 191
Analyseverfahren 45
Anfangsregel 42, 45, 48, 52
Anfügkategorie 18
Anwendbarkeit, potentielle __ von Derivationsanweisungen 83, 115
Anwendungskomponente 5, 62, 91f., 92, 95f., 189, 194
Anwendungskorpus 95
APPLY 95, 128f., 261f.

Basismodell, alternatives __ eines Lernalgorithmus 183-187
Basismodell eines Lernalgorithmus 3ff.
Basisregel 52
Basissegment 24
Beschreibungssprache, formale 74, 76-84
Bewertungsverfahren 46f.
BLCKSTS 102, 109-111
Blockierungsanweisung 103-107, 109-111, 146-148, 181
-, präfixorientierte 78f., 80, 192
-, suffixorientierte 78, 80, 192f.

C-Adäquatheit 45f., 46f., 48, 54f.
CHKCRPS 94, 101, 213-218

Dependenzgrammatik 32
Derivationsanweisung 79, 80
-, externe 94f.
-, interne 94
Derivationsrelation 38
DERIVE 96, 129-131, 263-266
DISCOV 94, 102-124, 219-243

Forderung, erste 84f., 135, 144-154
-, zweite 85, 135, 136f., 154-164
-, dritte 85, 135, 136f., 164-168
-, vierte 87, 135, 136f., 168-173
-, fünfte 87, 135, 173-175

FRODSET 95, 127, 257-260

Generalisierung 2, 3, 52f., 108, 110
GETOPTN 92, 99f., 204-212
Graph 1
Graphem 1

Häufigkeitsrangfolge von Substitutionsmustern 86f., 112f.

Impediens 79, 104, 105, 106, 109f.
INVERT 96, 132f., 269f.

Kapazität, prädiktive 73f., 75, 85f.
Kern 17f.
Kontextbedingung 77, 112-121
Kontextstring 77
Konversion 63
Künstliche Intelligenz 5f.
Kürzung 81

Lemmatisierung, automatische 177
Lernalgorithmen für die Morphologie 14f., 22-27, 57-60
- für die Phonologie 41-56
- für die Syntax 14
Lernen durch Analogieschluß 9
Lernen, inferentielles 9
-, Typen des s 9-11
- von Regeln 10
Lernkomponente 5, 62, 91, 92, 94f., 189, 191-194
Lernkorpus 62, 191
-, repräsentatives 43f., 85, 86, 164-167, 180
Linkssegment 24
LISTOUT 96, 132, 267f.
LLTMTCH 96, 133f., 271

Materialaufbereitungsverfahren 45
Modellcharakter 5f., 11, 88-90
Modifikationsverfahren 47f.
MONITOR 92, 98f., 198-203
Motion 89
- im Französischen 135f., 136, 137f., 154, 155, 157, 158, 161, 165f., 167, 168-172, 272-295

Negation englischer Adjektive 1ff., 116-120, 172f., 296-305
Nomen actionis 139
Nominalisierung deutscher Verben 32-36, 136, 138-143, 154, 155, 159, 160, 161, 166f., 167
Nullstring 14, 30

O-Adäquatheit, schwache 45f., 46f., 48f.
-, starke 46
Ordnung von Mengen von Derivationsanweisungen 81f., 115f., 193

Phonemisierung, automatische 40, 41-56
Pluralbildung englischer Substantive 75, 178f.
Präfigierung 69
PREPCRP 102, 103-109
PREPSTS 103, 123

PRISM
-, Anwendungskomponente von __ 95f.
-, Anwendungsmöglichkeiten von __ 6f., 176-180
-, Evaluierung von __ 144-175
-, Interne Struktur von __ 91-134
-, Lernkomponente von __ 94f.
-, Modellcharakter von __ 5f., 11, 88-90
- Quellprogramm 188-271
-, Unzulänglichkeiten von __ 181-183
-, Weiterentwicklungsmöglichkeiten von __ 181-183
Prüfverfahren 45f.

Quellstring 62

Rechtseindeutigkeit von Wortpaarlisten 71f.
Rechtssegment 24
Redundanz 45, 50, 84, 173-175
Redundanzfreiheit 84, 87, 104, 108, 110, 170f., 173-175
Regel, graphisch basierte 74-76
-, kontextsensitive 31, 44, 51, 77
-, kontextfreie 77
Regelerzeugung 52
Regelhaftigkeit von morphologischen Ableitungsbeziehungen 8f., 180
Regelkollision 53
Regellernen 10
Regelmenge, geordnete 29, 31, 44, 52, 53f., 81f., 115, 193
Regelstabilisierung 54
Resultatkategorie 18

Satellit 17
Satellitenklasse 18
Satzrelation Handlung 33, 34, 35
- Merkmalzuordnung 33, 34, 35
- Vorgang 33, 34, 35
- Zustand 33, 35f.
SBSTSTS 103, 111-123
Simulation derivativer Wortbildungsregeln 16-39
Spracherwerbstheorien, empiristische 90
-, rationalistische 90
Stamm 63
Stellungskategorie 18
Steueroptionen 100, 189f.
STMT 94, 102f., 123
STMTOUT 95, 124-126, 244-251
Substituendum 77, 107-109
Substituens 77, 107-109
Substitutionsanweisung 107-109, 111-123, 148-154, 181f.
-, kontextfreie 77, 121-123
-, kontextsensitive 77, 81, 112-121
-, präfixorientierte 76, 86, 193
-, suffixorientierte 77, 80, 193
Substitutionsbeziehung 14f., 40
Suprafigierung 70
Syntheseverfahren 45

Teilstring 30
-, echter 30
-, linker 30
-, rechter 30
TODSET 95, 126f., 252-256
Typen des Lernens 9-11

Überdiskriminierung 3, 8
Übergeneralisierung 2, 8, 10, 20, 53, 54, 163f.
Umsetzungszusammenhang, natürlicher 42

Verfahren, lexikonbasiertes 8f., 182f.
-, regelbasiertes 8f., 182f.

Wissensbasis 5, 62, 91
Wortpaarliste 62

Zeichenkette 14
Zielstring 62

Klaus W o t h k e

L e b e n s l a u f

Geboren: 17.8.1956	in Köln Vater: Friedrich Wothke (†), Dipl.-Kaufmann Mutter: Theresia Wothke (geb. Hauffe), Kaufmännische Angestellte
Staatsangehörigkeit:	Deutscher
Familienstand:	Ledig
Ausbildung und berufliche Tätigkeiten:	
1963 - 1966	Besuch der Ernst-Moritz-Arndt-Schule (= Grundschule) in Rodenkirchen (Kreis Köln).
1966 - 1975	Besuch des mathematisch-naturwissenschaftlichen Humboldt-Gymnasiums in Köln. Abschluß: Abitur.
1975 - 1977	Grundstudium in den Fächern Germanistik und Anglistik an der Universität Marburg/Lahn mit den Studienschwerpunkten Linguistische Datenverarbeitung, Linguistik des Deutschen und Linguistik des Englischen.
1977 - Mai 1981	Hauptstudium in den Fächern Kommunikationsforschung und Phonetik, Allgemeine Sprachwissenschaft, Logik und Grundlagenforschung an der Universität Bonn mit den Studienschwerpunkten Linguistische Datenverarbeitung, Linguistik des Deutschen, mathematische Logik und Phonetik. Abschluß: Magister Artium.
Juni 1981 - Sept. 1982	Zivildienst in der HNO-Universitätsklinik/Bonn. Tätigkeit: Programmierung und Wartung eines Signalanalysecomputers.
seit Okt. 1982	- Doktorand im Fachgebiet Linguistische Datenverarbeitung des Instituts für Kommunikationsforschung und Phonetik der Universität Bonn. - Studentische Hilfskraft in der Bonner HNO-Universitätsklinik. Tätigkeit: Entwicklung eines maschinellen Patientendatendokumentationssystems. - Studium des Faches Audiologie.
Weiteres: 1976 - 1980	Stipendiat der Studienstiftung des Deutschen Volkes.

Bonn, den 7.VII.1984

Klaus Wothke